Methods of Symmetric Cryptanalysis

Dmitry Khovratovich

Microsoft Research Redmond, USA

July 1, 2011

Introduction

Cryptography is the science of hiding information. It is now a part of the computer science formally, though first cryptographers appeared thousands years before the computer. The art of recovery of the hidden information, or *cryptanalysis*, appeared in the very beginning, and is still one of the most intriguing part of cryptography.

Cryptanalysis starts with a search for a weakness in a cryptosystem, for a flaw that was missed by its designer. An encrypted message must not reveal any information about its origin, so the cryptosystem must make it look as random as possible. Any mistake, any missed property may become a target for a cryptanalyst and a starting point for a compromise of the cryptosystem's security — a *break*.

This survey is devoted to the cryptanalysis of *symmetric primitives*. Historically, by a *symmetric encryption* we understand that all the parties have the same information needed for encryption and decryption, with block and stream ciphers as the most famous examples. A *block cipher* transforms a large block of data with an algorithm parametrized by a secret key. A *stream cipher* expands a secret key into arbitrarily long sequence, which is mixed with a data stream.

Hash functions convert a data string to a fixed-length hash value, which serves as an integrity certificate. Though hash functions do not encrypt, they are designed similarly to block ciphers. *Message authentication codes* (MAC) produce a hash value using a secret key, so they are between ciphers and hash functions. As a result, the cryptanalysis of hash functions and MACs employs methods that were initially developed for the analysis of block ciphers.

Ciphers, hash functions and MACs process arbitrarily long data streams, the access to which is sequential. This leads to the principle of an *iterative design*, where data is divided into blocks, and each block is processed by an algorithm with a fixed-length input. Such algorithms for hash functions are called *compression functions*. In contrast, by a *block cipher* we mean a primitive with a fixed-length input, which is used to encrypt arbitrary long data in a *mode of operation*.

We are primarily interested in the methods that are used in attacks on at least two different primitives. Cryptanalysis is often described as a cloud of non-related and dedicated attacks, which can be used only once. We introduce it in a more structured way.

Contents

Part I

Framework

Chapter 1

Block ciphers

A *block cipher* $E(K, P) \equiv E_K(P)$ is a bijective transformation (*encryption*) of an n-bit *plaintext* P under a k-bit *key* K, where k and n are the parameters. The result of the encryption is called a *ciphertext*.

In sections 1.1 and 1.2 we consider various attack goals and scenarios. For various scenarios we consider a *generic attack*, i.e., an attack that is universal for every cipher with the same dimensions. A cipher is secure as long as a generic attack is the best in terms of complexity.

Contents

1.1 Attack goals

Confidentiality violation

If a block cipher is used for confidentiality purposes, the most typical attack is the reveal of the hidden information. An attack procedure is formally divided into three phases. In the *offline phase of the attack* an adversary possesses only the description of the cipher, and is able to do precomputations. In the *online phase*, the adversary is given access to some plaintext-ciphertext pairs (*data*) and have specific time and memory resources.

In the *challenge phase* the adversary must recover the plaintext given access only to a ciphertext. We assume she has to find a secret key for that purpose, i.e. to mount a *key-recovery attack*.

If only a part of the key is found (*partial key recovery*), the attack is still considered dangerous, because the remaining part of the key is usually found with just a bit more sophisticated technique. Even if we find those key bits by exhaustive search, the total computational complexity is smaller than that of the exhaustive search.

Intuitively, a good cipher with a fixed secret key should behave like a random permutation. A widespread (though informal) notion of a *distinguisher* stands for an algorithm that provide evidence that a cipher behaves non-randomly.

A distinguisher serves not only as a certificate of a weakness, but also as a base for an advanced attack. A typical example is the following. Given a cipher, we collect as many plaintexts and ciphertexts as required by the distinguisher. Then we guess a part of the key so that the encryption process can be partly undone, and apply the distinguisher to a set of pairs $\langle$ encrypted plaintext, partly decrypted ciphertext$\rangle$. If the guess is correct, the distinguisher succeeds, otherwise it should fail.

In a differential attack (Section 3.2) the property is defined as a fixed difference in a pair of plaintexts (ΔP) or ciphertexts (ΔC). A differential distinguisher typically claims that the proportion of pairs with particular differences ($\Delta P, \Delta C$) is significantly higher than in a random permutation.

Security. In terms of the total computational complexity, the best key recovery attack is a simple exhaustive key search, which recovers a key with probability p after $p2^k$ cipher calls. Therefore, a block cipher is called secure against a key recovery attack, if the average computational complexity of the best attack is equivalent to 2^{k-1} or more cipher calls.

The complexity of the exhaustive search is concentrated in the online phase, but little precomputation is required. It is also possible to balance the complexity between two phases in tradeoff attacks [Hel80, Oec03, BMS05, BBS06].

Attacks on high-level constructions

Block ciphers are extensively used as building blocks in high-level constructions. There exist blockcipher-based pseudo-random generators, stream ciphers, MACs and hash functions. In some constructions, i.e. in compression functions, a key of the block cipher is not considered secret and fixed. Furthermore, an attacker might be able to observe and even change a key in an attack on the whole construction.

Clearly, what is undesirable from a block cipher depends on the construction where it is used. The construction is usually proved to be secure to an attack A, assuming that the underlying cipher is an *ideal cipher*, i.e. is a set of randomly chosen permutations. An ideal cipher with n-bit plaintext and k-bit key implements 2^k (out of $(2^n)!$) permutations over Z_2^n. An adversary is given access to both encryption and decryption devices, and a key, a plaintext, and a ciphertext can be chosen by him. E.g., for the Davies-Meyer mode any universal (working for any cipher) collision-finding algorithm fails with high probability if it makes less than $2^{n/2}$ queries to both devices [BRS02].

As a result, a property that is unlikely to find for an ideal cipher, is a potential violation of the whole construction. However, due to complicated nature of some properties, it is hard to compare a "non-randomness" attack with generic attacks.

Complexity issues

The applicability of an attack depends not only on the type of freedom that the adversary possesses in the online phase, but also on the total computational complexity of the attack. A practical attack on a cipher clearly implies that it should not be used for confidentiality purposes.

Though practical attacks are rare, users prefer to be aware of any weaknesses in a cipher that may lead to an attack in future. As a response to these practical needs, the cryptographic research community considers purely theoretical attacks valid if they demonstrate an undesirable property of a cipher. Here by a *theoretical attack* we mean an attack that can not be mounted in the real world due to high time or data complexity, memory requirements, or the protocol restrictions.

1.2 Attack scenarios

The attack scenarios are grouped according to the power of an adversary in the online phase. When the adversary is restricted in encryption, decryption or viewing results, we call this a *data treatment* parameter. When we restrict the adversary's influence on the key K, it is a *key treatment* parameter.

1.2.1 Data treatment

In practice an adversary is limited in the access to the results of encryption and decryption. According to these limits, attacks are classified as follows.

The weakest adversary has access only to encrypted messages and to statistical information about the plaintext: the frequency of letters, the length of the message, etc.. Such an attack is called a *ciphertext-only* attack. A few ciphertext-only attacks exist [BK98a], but widely used ciphers are rarely affected.

If an adversary has access to both plaintexts and ciphertexts, it is a *known-plaintext attack*. If he is also able to encrypt arbitrary plaintexts, it is a *chosen-plaintext attack*. Analogously, if he can decrypt arbitrary ciphertexts, it is a *chosen-ciphertext attack*.

We also consider adaptive and non-adaptive attacks. In the *adaptive* attack an adversary is able to choose his encryption or decryption queries during the online phase, while in the *non-adaptive* attack he prepares the queries in advance.

1.2.2 Single secret key

Regarding the key treatment, the single secret key scenario is the most popular. It is a typical scenario for attacks on block cipher as an encryption primitive. A cipher is considered *secure in the single secret key scenario*, if there is no key recovery algorithm faster than the exhaustive search, i.e. with computational complexity smaller than 2^{k-1} encryptions on average.

1.2.3 Related keys

In the related-key attack an adversary is able to decrypt and encrypt not only under the key K, but also with keys $f_1(K), f_2(K), \ldots, f_r(K)$, which are called *related keys*. The *relation mappings* f_i are chosen by the adversary. The first related-key attacks dealt with simple mappings: rotations [Bih94] and bit flips [KSW97]. Recent attacks on AES [BK09] exploit dedicated relations, where the fixed difference is applied to subkeys, but not to original keys. These attacks were later called related-subkey attacks [BDK+10].

Related-key attacks are used in the attacks on the protocols that use ciphers as a building block and derive their keys in an unsecure way [TWP07]. Related-key attacks, in contrast to open-key attacks (Section 1.2.4), recover a secret parameter of the protocol.

Definition problems. It is hard to formalize the security against related-key attacks. The problem is that several key relations admit trivial attacks with low complexity, which apply to any cipher [BK03]. Even worse, for any given cipher it is easy to define a relation that leads to a trivial attack on this particular cipher, e.g. the relation $f(K) = E_K(1)$. To rule out this class of attacks, one may try to restrict the set of relations that are "admissible", e.g. to forbid the relations that employ the internal operations of the cipher. The disadvantage is that such a restriction also kills already approved related-subkey attacks like [BKN09].

Another idea is to consider related-key attacks only as an application to high-level constructions that use a cipher as a building block. Therefore, each related-key attack enlarges the set of protocols, which are insecure if instantiated with a particular cipher.

Number of keys. The number of related keys used in the attack is also an important parameter. Earlier related-key attacks on AES [BDK05b, KHP07] employ 64 keys with a simple relation in order to get a pair with a desirable relation. We notice that the generic time/memory/data/key tradeoff attack [BMS05] recovers one of T keys with complexity $2^n/T$ for an n-bit block cipher.

1.2.4 Open-key model

The key is not secret in the recently appeared *open-key model*, where the key and the plaintext are almost equivalent inputs. So far, the most common analysis in the open-key model is to construct so called *evasive property*, which explicitly links plaintexts and ciphertexts (see examples in Section 2.1). This property must be difficult to find for an ideal cipher as well as for other dedicated ciphers, but not for a particular cipher. It can also serve as a certificate of "non-ideal' behavior' of the cipher.

The open-key model can be split into two sub-models. The first is the *known-key model*, where a key is known but fixed in advance. A cipher is now a permutation, which should behave as a random permutation. The second type is the *chosen-key model*, where both a key and a plaintext can be chosen by the adversary. In the chosen-key model a cipher is viewed as a set of independently chosen permutations and is compared to an ideal cipher. The chosen-key model is the closest to the model where hash functions are investigated.

Chapter 2

Hash functions

Contents

Hash functions in cryptography are primarily used for data integrity and message authentication purposes. They map a string M (*message*) of arbitrary length into a fixed-length output $h = H(M)$, referred to as a *hash value*, or simply *hash*.

Since a hash function must process inputs of arbitrary length, it is typically constructed as an iteration of a *compression function* (further denoted by E) with a fixed-length input and output. Probably, the first cryptographic scheme of this kind was proposed by Rabin [Rab78]: the message M is split into blocks $m_0||m_1||m_2\cdots m_k$, which are keys of a block cipher (DES in Rabin's scheme). The first block and a pre-fixed *initial value* (IV) are compressed into the first chaining value

$$CV_1 = E_{m_0}(CV).$$

Then we define

$$CV_i = E_{m_{i-1}}(CV_{i-1}), \qquad \text{and } H(M) = CV_{k+1}.$$

Many dedicated hash functions (e.g., the MD-family) are not based on block ciphers formally, but incorporate many blockcipher-specific ideas, particularly the message schedule (key schedule in block ciphers). The main class of hash functions that do not use any schedule is sponge functions [BDPA07] and their derivatives [BDPA06, KRT07], where a message block is simply injected into the chaining value by XORing or substitution and is not used elsewhere:

$$IV_i = F(IV_{i-1} \oplus m_{i-1}), \qquad H(M) = \text{FinalTransformation}(IV_{k+1}).$$

A variant of hash function, where m_i is concatenated with the chaining value, and then the output is truncated, is actually a modification of the sponge construction:

$$IV_i = \text{Trunc}(F(IV_{i-1}||m_{i-1})), \qquad H(M) = \text{FinalTransformation}(IV_{k+1}).$$

The security requirements to a hash function depend on the application where it is used. In the following sections we discuss the most popular applications and attack scenarios.

2.1 Attack goals

In this section we discuss the goals of attacks on hash functions. We use an application-based approach, i.e. we consider the most popular applications and formalize attacks that violates their security.

Forgery

Hash functions are widely used in signature schemes, where digital signatures are applied to the hash of the string:

$$\text{Signature} = S(H(M)).$$

If an adversary finds an alternative message M' producing the same hash value $H(M)$, this may compromise the scheme. A pair of inputs (M, M') that are mapped to the same value is called a *collision*.

If M is fixed in the attack, the adversary has to find a *second preimage*, which is a harder problem. Since the domain of hash function is larger than its range, both collisions and second preimages are unavoidable in the information-theoretic sense. As a result, the security of a hash function is formulated in the computational-complexity sense:

- It should be computationally infeasible to find two messages resulting into a same hash value (*collision resistance*): $H(M_1) = H(M_2),\ M_1 \neq M_2$;

- It should be computationally infeasible to find the second message giving the same hash (*second-preimage resistance*): given M find M' such that $H(M) = H(M')$.

It is easy to prove that collision resistance implies second-preimage resistance.

An authentication scheme can be also based on a *message authentication code* (MAC), which computes a hash of the message using a key K:

$$\text{MAC}_K(M) = F(K, M),$$

where F is a hashfunction-based transformation. The key must be chosen by the parties in advance.

The key recovery and a collision for MAC are natural threats for an authentication scheme. Also, a MAC of $(M_1||M_2)$ should not be computed from a MAC of M_1 nor M_2. Otherwise an adversary is able to predict the code having no key (*length-extension* property):

$$\text{MAC}(M_1||M_2) \overset{?}{=} F(\text{MAC}(M_1), M_2).$$

Recovery of secret information

Hash functions are widely used for the password protection. An application stores not passwords themselves, but hashes of the passwords:

$$\text{PasswordList} \leftarrow H(\text{Password}).$$

As a result, if the hash file is captured by an adversary, she has to invert a hash function in order to get a valid password. In other words, the adversary is given an image I, and she has to find a *preimage* to P such that $H(P) = I$. We note that a preimage does not have to be an original password.

Therefore, for a good hash function it should be computationally infeasible to recover a message from the hash (*preimage resistance*): given v find M such that $H(M) = v$.

Proof of knowledge

Hash functions are used in *proof of knowledge protocols*, where a participant proves the knowledge of some information $\mathcal{Q}$ by committing the hash of this information and disclosing the preimage in the last step of a protocol. The participant has some freedom in choosing the message by taking arbitrary nonce r or adding garbage bits:

$$\mathcal{Q} \rightarrow \text{Message } M \rightarrow H(M) = I.$$

An adversary who does not possess the knowledge but wants to forge the proof, has to find a valid preimage to I which contains $\mathcal{Q}$. However, since the adversary is able to choose the committed value, she might be able to compute it in a way that simplifies the search. Therefore, the attack is weaker than regular preimage attack. A generic attack with complexity slightly higher than $2^{n/2}$, which works for any iterative hash function, was demonstrated in 2006 [KK06]. However, there is little evidence on how to attack a single call of a compression function.

Violation of randomness

Hash functions are also used in pseudo-random generators and key derivation protocols. These applications assume that the output of a hash function has random behavior, i.e., the hash function should simulate a randomly chosen function. Therefore, any *non-randomness* property of the hash function may violate the security requirements of the protocol.

However, the theory of cryptography does not answer the question which function is not random, it can only prove inability to distinguish a member of a large family from a randomly chosen function. While we can theoretically do this for a cipher as it is parametrized by a key, we can not do this for a given hash function with no secret input, nor for a cipher with a fixed key. What we can do is to demonstrate a property that should not be present in an ideal cipher (or a hash function). Concrete examples are given in Section 2.2.

Violation of security proofs

The security level of a hash function is often based on the security of its low-level components. For example, some modes of SHA-3 candidates [BCCM+09, Riv08, FLS+08] are provably secure assuming some properties of the underlying primitive. Therefore, any attack that violates such property is a reasonable object for cryptanalysis. A hash function can be also based on one or several permutations, which are compared with a random permutation.

Secutity of some hash functions is provided only by the finalization procedure, which does not operate on a message input [KRT07, BDPA06]. We can say that such primitives are explicitly based on weak blocks.

Comparison of designs

A rather weak attack on a hash function is still relevant if it allows to compare the security of different designs that are equally resistant to basic attacks such as key recovery or collision search.

Iterative block ciphers can be compared by the number of broken rounds. However, there is no consensus on which security margin is sufficient, and how to compare ciphers with different number of rounds. The same approach fails for hash functions, which is clear since many competitive designs have been proposed for the SHA-3 competition.

A new tendency is to compare the strength of attacks that can be applied to the original, non-reduced version of a primitive. As a good hash function and a block cipher should behave as a random function (or permutation), we can measure how far the investigated primitive is from a random function. Again, there is little evidence on how to define a "non-randomness" property properly.

2.2 Attack scenarios

In this section we classify attack scenarios by the constraints imposed to an adversary.

Fixed output difference

An adversary is given a difference D and has to find

$$M \neq M'" \; H(M) \oplus H(M') = D.$$

If $D = 0$ we call the pair (M, M') a collision pair. A generic collision attack is widely known as the *birthday attack*. The *birthday paradox* states that among 23 randomly chosen people there likely to exist two with the same birthday. More generally, randomly chosen $\sqrt{2N}$ (out of possible N) elements are unlikely to be all different, because there are N possible pairs. Therefore, as first was noted by Yuval [Yuv79], one can generate a collision for an n-bit hash function by hashing and sorting about $2^{n/2}$ messages. There also exist memoryless modifications of this method, which are only marginally slower (Section 5.1).

There are several weaker definitions of the collision, which aim to find a weakness in a primitive compared to a random function. Two messages produce a *near-collision* if they collide only in a part of the hash value. A t-bit near-collision refers to messages colliding in t bits. Clearly, the birthday attack would need $\approx 2^{t/2}$ evaluations of the hash function. A near-collision in a compression function might become a regular collision in a hash function, if the output is modified with a kind of a feedforward. The application of this principle to multi-block collisions is described in Section 8.4.

Since any, even accidentally found, collision violates the property of collision resistance, the definition of the latter seems to be incomplete. However, the only way to formalize it further is to introduce a concept of a family of hash functions [RS04] such that a collision-search algorithm would succeed for any family member. So far, there is no consensus on how to put designs like SHA-2 into this concept.

There is a link between the collision resistance of compression functions and hash functions. Merkle and Damgård [Mer89,Dam89] proved that if the compression function is collision-resistant, and the last message block contains the message length, then the resulting hash function is collision-resistant. Therefore, it is reasonable to attack compression functions, and afterwards extend the results to the whole hash function. However, in the attack on the compression function an adversary have more freedom, since there is no fixed initial value. As a result, attacks on compression functions are relatively hard to convert to full-scale attacks.

Fixed output

An adversary is given I and has to find M such that $H(M) = I$. This is a typical preimage search problem. A good hash function with an n-bit output should be resistant to preimage attacks with complexity significantly less than 2^n. This is what we expect of a random function with the same output size.

If the hash function is based on the iteration of compression functions $E(IV, M)$, we distinguish between preimages to the compression function and to the hash function itself. A preimage (IV', M) for a compression function, if $IV' \neq IV_0$, is called a *pseudo-preimage*. There are several ways to compute a preimage for the n-bit hash function given many pseudo-preimages (Section 5.1), keeping the total complexity below 2^n.

A *second preimage* search, when we know an $M' : H(M') = I$, is typically easier than the preimage search, since other techniques (mostly differential based) can be applied. Another advantage is that a primitive can be attacked from both ends: an adversary is able to construct a valid computation from the very beginning while in the preimage search he is restricted to the given hash value.

Restrictions on input

The restrictions on input values are additional constraints used in the previous scenarios. A typical hash function has only one admissible initial value IV_0, which is a significant restriction on the attack. In the compression function setting we do not have a restriction on inputs formally, but we can distinguish different types of attack by the amount of control over the IV input. The following taxonomy is used in collision attacks:

Initial value 1	Initial value 2	Name
IV_0	IV_0	Collision
IV'	IV''	Pseudo-collision
IV	IV	Semi-free-start collision

where IV, IV', IV'' are pairwise different and not equal to IV_0.

All these versions of collisions can be combined, resulting in a pseudo-near-collision, a free-start collision, etc.. If there is an algorithm that finds this type of collisions faster than for a random function, it is considered a weakness.

It is also worth to consider collision attacks, where some parts of the message block (or previous message blocks) are restricted. This is quite common in attacks on real protocols, such as X.509 [SLdW07, SSA+09] and PostScript [DL05]. Another application is the search for a meaningful collision, where both colliding messages contain valid information blocks.

Combined input/output relations

The most "non-random" (or evasive) properties introduced so far are primitive-independent. Informally, we should not figure out which cipher is attacked just by looking at the property, so these attacks are considered valid and universal. Certainly, the most interesting properties are those convertible to real attacks such as key recovery or collision.

The evasive properties, constructed in recent papers, correspond to following primitive-independent problems:

- Given a permutation p, find a k-sum to zero of $(x||p(x))$ [KR07].
- Given a permutation p and constraints on both input and output space, find x such that both x and $p(x)$ satisfy these constraints [Kho09b].
- Given a permutation, try to find t input pairs such that their difference is zero in i bits, and difference in their outputs is zero in j bits [GP10].
- Given a block cipher, try to find t input $\langle$key, plaintext$\rangle$ pairs that have and produce equal difference [BKN09];
- Given an n-bit function, try to find t input pairs such that the corresponding output differences belong to a vector space of dimension $k < n$ [LMR+09].

The papers referenced above also provide with a description of a generic algorithm and its complexity.

One of these relations, the so-called CICO problem, deserves a separate paragraph. We consider the problem of finding a relation

$$f(x) = y \; , \tag{2.1}$$

such that the first q bits of x and y are fixed to some pre-determined value (e.g., to zero), $q \leq \min(n, m)$. Bruteforce search, which works for any f, requires about 2^q computations of f. This bound holds as well when $n = m$ and when f is invertible.

One expects that for a good transformation, this problem should have the same workload.

A more general problem has been proposed by the designers of KECCAK [BDPA09]. Assume that $n = m$ and define $X \subseteq \{0,1\}^n$ as a set of possible inputs and $Y \subseteq \{0,1\}^n$ as a set of possible outputs. Then find a solution to Equation (2.1) with $(x, y) \in X \times Y$.

Part II

Methods

This part describes the state of the art of modern cryptanalysis, with concentration on block ciphers and hash functions. In each chapter we introduce an idea and then demonstrate how it can be converted to one of attacks from Part I.

Chapter 3 is devoted to probabilistic patters —properties that hold in nonlinear transformations with some probability. Famous linear and differential cryptanalysis methods fit this framework. Chapter 4 describes methods that are specific for primitives that work with small blocks: bytes and words. We show how to use this level of abstraction to obtain more efficient properties than in bit-oriented primitives. Chapter 5 explains how to exploit the fact that the output of the transformation is shorter then its input due to some of form of schedule or injection. Here non-bijectivity of transformations play an important role in the analysis.

Then we abstract from the details of transformations and consider primitives on the high level. Chapter 6 demonstrates how to decompose a primitive into smaller parts, so that for which part the most efficient pattern is applied. Chapter 7 discusses how to change and exploit the high-level structure of a primitive, and how to derive structure-dependent invariants. Finally, in Chapter 8 we explain how to use all the methods in the most efficient way, and introduce several tricks that speed up an attack.

Chapter 3

Analysis of nonlinear transformations

This chapter is devoted to the statistical analysis of nonlinear transformations. The concept is to find properties that hold with reasonably high probability. These properties can be used in various scenarios: key recovery in block ciphers, collision attacks on hash functions, and nonrandomness certificates. They can be used alone or in conjunction with other properties.

Statistical analysis works with *probabilistic patterns*, which are defined for a function f, an input property $\mathcal{A}$, and an output property $\mathcal{B}$. We say that a pattern holds with probability p if

$$\mathbb{P}\left[\mathcal{A} \xrightarrow{f} \mathcal{B}\right] = p.$$

Linear and differential cryptanalysis are widely known examples of probabilistic patterns and will be discussed in the following sections. We introduce them separately for more concrete explanation, though they may be fit into a single framework like in [Vau96].

Contents

3.1 Linear cryptanalysis

The idea of linear cryptanalysis is to approximate non-linear components with linear functions. A linear approximation is a linear relation between input and output bits of a transformation. Formally, given a function f, we find boolean vectors (*masks*) u and v such that

$$u \cdot X = w \cdot f(X)$$

holds with probability $1/2+\varepsilon$ for reasonably high $|\varepsilon|$. Here $\cdot$ is the inner dot-product. If $f = f_K$ is a keyed function, the relation is supposed to involve bits of K as well:

$$u \cdot X \oplus w \cdot f_K(X) = v \cdot K. \tag{3.1}$$

In the attack on DES [Mat93] the S-box S_5 was approximated with relation $(0,0,0,1)\cdot X = (1,1,1,1)\cdot S_5(X)$, which holds with probability $1/2-5/16$. As a result, the following approximation holds for the round function F:

$$X_{15} \oplus F_K(X)_7 \oplus F_K(X)_{18} \oplus F_K(X)_{24} \oplus F_K(X)_{29} = K_{22},$$

where indices refers to bits.

A natural question is how to derive equations of form (3.1) for the whole primitive. First, we find approximations for each round, such that masks (u^i, v^i, w^i) can be connected:

$$u^{i+1} = w^i,$$

i.e. the output bit mask coincides with the input bit mask of the next round. The sequence

$$u^1 \xrightarrow{v^1} u^2 \xrightarrow{v^2} \cdots \xrightarrow{v^r} w^r$$

is called a *linear characteristic*, or a *linear trail.* A set of characteristics with common plaintext and ciphertext masks is called *a linear hull* (analogue to a differential). The bias of the approximation is given by the following observation.

Proposition 1 ***(Piling-up lemma)*** *[Mat93] Let $X_1,\dots,X_n$ be n statistically independent Boolean random variables with $\mathbb{P}[X_i=0]=1/2+\varepsilon_i$. Then*

$$\mathbb{P}[X_1 \oplus X_2 \oplus \cdots \oplus X_n = 0] = \frac{1}{2} + \frac{1}{2}\prod_i (2\varepsilon_i).$$

Under some independence assumptions, we sum all the approximations and derive that

$$\bigoplus_i \left(u^i \cdot A^i \oplus w^i \cdot A^{i+1} \oplus v^i \cdot K^i\right) = 0 \quad \Longrightarrow$$

$$\Longrightarrow \quad \bigoplus_i v^i \cdot K^i \oplus u^1 \cdot A^1 \oplus w^R A^{r+1} \quad \Longrightarrow \quad u^1 \cdot P \oplus w^R C = \bigoplus v^i \cdot K^i = \overline{v} \cdot \overline{K},$$

holds with probability $1/2(1+\prod(2\varepsilon)^i) = 1/2+\varepsilon$. Here K^i is the i-th subkey.

Formalization. More formal approach to linear cryptanalysis deals with the *correlation* C of function f instead of probabilities:

$$C(f) = \frac{\#\{x \,|\, f(x)=0\} - \#\{x \,|\, f(x)=1\}}{2^n},$$

where n is the dimension of the domain of f.

Then we either approximate $\overline{v}\cdot\overline{K}$ with $u\cdot x \oplus v \cdot f_K(x)$ (procedure called Algorithm 1 due to [Mat93]), or use the imbalance of $u \cdot x \oplus v \cdot f_K(x)$ as a statistical distinguisher (Algorithm 2). The complexity of both variants depends on the correlation of the latter value:

$$C_f(u,w) = C\left(u \cdot x \oplus w \cdot f_K(x)\right),$$

which is also called the *potential of the linear hull* (u,w). Evaluation of the potential is difficult, since all trails that start from u and end with w (denoted by $\mathcal{U}(u,w)$ or simply $\mathcal{U}$) contribute to the potential:

$$C\left(u \cdot x \oplus w \cdot f_K(x)\right) = \sum_{\mathcal{U}(u,w)} \prod_r C\left(u^r \cdot x \oplus u^{r+1} G(x, K^i)\right),$$

where G is the round function. Therefore, the potential is key-dependent. Ciphers like DES and Serpent have a single dominating trail whose correlation is the largest term in the sum. As a result, these ciphers are suitable for Algorithm 1 where key bits are derived explicitly from the approximation.

In contrast, ciphers like PRESENT yield many trails with similar correlation value, which makes a straightforward application of Algorithm 1 difficult (see [RN11] as an example). On the other hand, Algorithm 2 needs the potential value evaluating. We have to use the property specific for key-alternating ciphers:

$$C_f(u,w) = \sum_{\mathcal{U}} (-1)^{s(U,K)} |C_f(U)|, \tag{3.2}$$

where the *trail correlation* $C_f(U)$ is independent of the key and is computed as a multiplication of the round correlations (also key-independent). Therefore, linear trails add or subtract to the total correlation, depending on the key [DR02] (see also discussion in [Lea11] on this topic). If the trail correlations are equal, the potential distribution can be approximated with the normal distribution [Lea11]. If the round keys are assumed independent, we have the following expression for the potential value averaged over all keys [DR02, p.106]:

$$\mathbb{E}\left[(C_f(u,w))^2\right] = \sum_{\mathcal{U}} (C_f(U))^2 . \tag{3.3}$$

If the round keys are dependent and linear, each pair of trails contributes to the expected potential in a separate term as soon as the signs $(-1)^{s(U,K)}$ are equal for all keys, which occurs mainly if the trails are abundant [DWS10]. For the non-linear key schedule the similar condition is less restrictive but still unlikely [DR02, p.108]. As a result, the approximation (3.3) is valid in the most of cases.

Extensions An extension to this idea is the use of multiple linear hulls simultaneously [JR94, BCQ04], which was later generalized to multidimensional linear approximations [HN10, HN11]. In the latter attack we approximate an m-bit function f and use $2^m - 1$ correlations $C(af)$, $a \in Z_2^m$ for the discrete probability distribution table, a generalization of the regular correlation. The properties we have presented earlier generalize to the multidimensional linear cryptanalysis, though the attack procedure becomes more complicated. The data complexity is determined by the capacity Cap:

$$Cap(f) = \sum_{a \in Z_2^m, a \neq 0} C^2(af).$$

The method led to the best attack on PRESENT [Cho10].

Another extension is the use of arbitrary functions instead of a linear sum. The partitioning cryptanalysis [HM97] and non-linear approximations [KR96, SK98] are examples of this idea. However, the use is still limited due to lack of good approximations and small advantage over the basic attack.

3.2 Differential cryptanalysis

The idea of the differential cryptanalysis is to consider the $\oplus$-*difference*

$$\Delta^{\oplus} P = P_1 \oplus P_2 \tag{3.4}$$

between plaintexts P_1 and P_2, and its propagation through nonlinear and linear transformations of a primitive. Besides the $\oplus$-difference, the $\boxplus$-difference is also useful for the analysis of primitives with modular addition (Section 3.3). Further generalizations of difference are mostly theoretical.

The differential over a transformation F is a probabilistic pattern that maps an input difference Δ_I and an output difference Δ_O:

$$\Delta_I \xrightarrow{F} \Delta_O.$$

The *differential probability* (DP) is the number of ordered pairs with input difference Δ_I and output difference Δ_O divided by the total number of pairs with difference Δ_I:

$$\mathrm{DP}_F(\Delta_I, \Delta_O) = \#\{\{x, y\} \mid x \oplus y = \Delta_I \text{ and } F(x) \oplus F(y) = \Delta_O\}/2^n,$$

where n is the output length.

The first differential attacks on iterative primitives dealt with *characteristics* (also called *trails*) — connected short differentials:

$$\begin{cases} \Delta_1 \overset{f_1}{\to} \Delta_2; \\ \Delta_2 \overset{f_2}{\to} \Delta_3; \\ \dots \\ \Delta_{k-1} \overset{f_{k-1}}{\to} \Delta_k. \end{cases} \quad \Longrightarrow \quad \Delta_1 \overset{f_1}{\to} \Delta_2 \overset{f_2}{\to} \dots \overset{f_{k-1}}{\to} \Delta_k.$$

One can omit restrictions on the intermediate differences and get a *differential* over a whole transformation: $\Delta_1 \overset{F}{\to} \Delta_k$.

The differential probability should be high enough to demonstrate a design weakness. For a random function the number of pairs following a differential is a random variable with the binomial distribution (in some cases can be approximated by the Poisson distribution) and with mean 2^{n-m}, where n is the input length and m is the output length [DR05]. Since the number of ordered pairs with fixed difference is 2^n, a differential probability is expected to be around 2^{-m}. Therefore a differential with a higher probability is a potential weakness, because a random mapping is unlikely to have high probability for the same differential.

However, the probability of a differential and even of a characteristic is quite hard to compute even for iterative transformations, though there exist good estimates under reasonable assumptions. Intuitively, a characteristic probability should be close to the multiplication of the probabilities of its internal differentials (so called *expected differential probability*, or EDP). In practice, this assumes independence between the inputs of the round function, which is not the case for many ciphers, and is hard to prove for the others. The situation is worse for iterative block ciphers, where each round gets a portion (or a function) of a key as an input. The differential probability, evidently, depends on the actual key value.

The best way to avoid the key influence is to consider *key-alternating ciphers*, where the round key is XORed to the internal state before each application of the round function, and the round function does not use the key. If round keys are independent, then DP = EDP for any fixed key (the same result holds in the *Markov cipher theory* [LM91]), otherwise the differential probability is a stochastic variable whose distribution has a mean equal to EDP [DR05]. For several old hash functions, such as the SHA family, the computation of the differential probability is hard, and the practice often compromises theoretical estimations.

3.3 Primitives with modular additions

Modular addition is a widely used operation in software-oriented primitives. The addition is fast in software, and is one of the fastest non-linear operations available to a designer. Though S-boxes are better in terms of nonlinearity, they are slower in the implementation.

Most significant bits in the addition arguments do not influence least significant bits of the result, which makes the diffusion unbalanced. As a countermeasure, designers combine addition with rotation or shift in order to balance the diffusion. Several primitives involve only three operations: addition, rotation and XOR, so they are called *ARX primitives*.

Most of analysis of addition-based systems has been done in the framework of differential cryptanalysis.

3.3.1 Additive differentials

If the modular addition is the only nonlinear (over $\mathbb{F}_2$) operation in a primitive, it is natural to consider modular differences instead of XOR differences. The modular difference passes the addition with probability 1, but passes the XOR non-deterministically. Also, several attacks on the ARX primitives specify characteristics on the bit level, where additive differentials are less suited. It was demonstrated that the analysis with additive differentials benefits rather from non-symmetric treatment of arguments than from the properties of the modular addition [CR06].

The bit level and the non-symmetric treatment of arguments were incorporated in a *universal bit condition* — a two-argument boolean function $f(x, y)$, that outputs 1 if (x, y) is good pair for the difference propagation. Then every bit condition in the differential trail can be expressed with this function.

3.3.2 Linearization

If the $\oplus$-difference has low weight, it propagates through addition with relatively high probability. This property leads to the idea of constructing differential trails with *linearization* — replacement of all additions with XORs, and computing difference propagation in the linearized primitive. The probability of the resulting trail is determined by the total weight of differences entering former additions.

Let us explain how to choose the input difference. The internal variables in the linearized version are just linear functions of the input. Therefore, the internal variables fulfill the equations of a linear code, and low-weight conditions on the variables can be translated into a low-weight condition on the codeword. The search for a low-weight codeword is a hard problem in the generic case, though several probabilistic algorithms exist [Leo88, Ste88, CC98], and they were applied in real attacks [PRR05, MP05].

3.3.3 Tools

Types of differences

Let us abbreviate the modular difference $(x - x')$ by $\Delta^+ x$, and the $\oplus$-difference $(x \oplus x')$ by $\Delta^\oplus x$. Since there is no bijective mapping between these types of differences, we introduce *signed bitwise difference*:

$$\Delta^\pm x = \Delta^\pm(x, x') = (x_{n-1} - x'_{n-1}, \ldots, x_0 - x'_0).$$

The next theorem links modular and $\oplus$-difference.

Theorem 1 ([Dau05]) *Let $\Delta^\pm x$ be the signed bitwise difference between two elements of $\mathbb{F}_2^n$. Then the $\oplus$-difference $\Delta^\oplus x$ and the modular difference $\Delta^+ x$ are uniquely determined.*

Differential analysis: xdp$^+$

We consider the differential properties of modular addition ($\boxplus$, see also [LM01] and the extensive treatment in [Wal03]). The probability that $\oplus$-differences α and β are transformed by $\boxplus$ to the $\oplus$-difference γ, is defined in the following way:

$$\mathrm{xdp}^+(\alpha, \beta \ \to \ \gamma) = \mathbb{P}_{x,y}\left[\big((x \oplus \alpha) + (y \oplus \beta)\big) \oplus (x + y) = \gamma\right].$$

There is a simple condition whether the xdp$^+$ is equal to 0:

$$\text{for some } i \in [0; n-1] \qquad \alpha_{i-1} = \beta_{i-1} = \gamma_{i-1} \neq \alpha_i \oplus \beta_i \oplus \gamma_i.$$

Here $\alpha_{-1} = \beta_{-1} = \gamma_{-1} = 0$. If the differential is "good", then let q be the number of positions $i \neq n-1$ where the triple $(\alpha_i, \beta_i, \gamma_i)$ has both one and zero. Then

$$\mathrm{xdp}^+(\alpha, \beta \ \to \ \gamma) = 2^{-q}.$$

Differential analysis: adp$^+$

The situation is more complicated for $\oplus$. If the arguments x and y of $\oplus$ have additive difference α and β, then the probability that $x \oplus y$ has difference γ is defined as follows:

$$\mathrm{adp}^{\oplus}(\alpha, \beta \ \to \ \gamma) = \mathbb{P}_{x,y}\left[\big((x + \alpha) \oplus (y + \beta)\big) - (x \oplus y) = \gamma\right].$$

Proposition 2 *[Wal03] There exists an algorithm with complexity $O(n)$ that computes adp$^{\oplus}$.*

The idea is to map each triplet $(\alpha_i, \beta_i, \gamma_i)$ to a matrix of constant size so that the value of adp$^{\oplus}$ can be computed by multiplication of n such matrices.

One may also apply the graph approach to benefit from the sparsity of the matrices and mitigate the memory issues possibly caused by the size of matrices. The graph approach is based on the concept of S-functions [MVCP10]. An S-function accepts n-bit words $a_1, a_2, \ldots, a_k$ and produces an n-bit word b and a list of states $\{S[i]\}$ in the following way:

$$(b_i, S[i+1]) = f(a_1[i], a_2[i], \ldots, a_k[i], S[i]), \quad 0 \leq i \leq n-1, \quad S[0] = 0. \tag{3.5}$$

The functions f are not necessarily the same, but the states $S[i]$ must have constant size. The modular addition is represented by a simple S-function, where b is the sum, and $S[i]$ is the carry produced at position $i-1$.

To compute adp$^{\oplus}$, we first represent a pair of additions as a single S-function with $k = 4$ and $S[i]$ consisting of three carries $c_1[i], c_2[i], c_3[i]$ that occur in the first output, in the second output, and in the output difference, respectively. The output b is the modular output difference. Then we build a graph with $8n$ vertices, each representing a possible value of $S[i]$ (eight combinations for each of n bits). Next, for each i we connect vertices $S[i]$ with $S[i+1]$ by edges according to Equation (3.5) and taking the edges whose b_i is equal to the desired γ_i. Finally, we compute the number of paths from $S[0]$ to $S[n]$, which is the number of inputs producing the output difference γ, and divide it by 4^n to derive adp$^{\oplus}$.

This approach can be adapted for computing the number of possible differences. It can be also reformulated in terms of non-determinite automata [LT11].

3.4 Algebraic attacks

In this section we introduce attacks that are based on the algebraic representation of a primitive.

3.4.1 High-order differentials

The $\oplus$-difference (Eq. 3.4) is closely related to the notion of a derivative in the continious analysis. Therefore, it is natural to consider high-order differentials as an application of the high-order derivative. A *k-order derivative* $\Delta_{\delta_1,\delta_2,\ldots,\delta_k}$ of a function f is the sum

$$\bigoplus_{c_i \in \{0,1\}} f(x \oplus c_1\delta_1 \oplus c_2\delta_2 \oplus \cdots \oplus c_k\delta_k).$$

As for continuous functions, a k-order derivative of a boolean polynomial of degree $< k$ is constant.

In contrast to the regular differential, which works with pairs, we iterate and compare 2^k states. However, the condition $\Delta_{\delta_1,\delta_2,\ldots,\delta_k}(f) = c$, being imposed on the inputs, is too weak and can not be traced with high probability through several non-linear transformations. As a result, the high-order differentials are not used as a statistical pattern. Instead, a high-order derivative is computed only once for the output of the whole transformation. If the transformation has low algebraic degree, a derivative of reasonably high order may provide a distinguisher. Original paper by Knudsen [Knu94] dealt with the Feistel ciphers, whose round function is a quadratic polynomial.

Apparently few real primitives have round functions, which are expressed by low-degree polynomials. As a result, high-order differentials are rarely applied (see attacks on MISTY1 [TSSK08] and Camellia [HSK02]). The property that a particular set of states sums to a constant has been also used in the Square/integral/multiset attack (Section 4.1), where it appears as a distinguishing property (the weakest one) of a permutation. However, in the multiset attack the origin of this property is different — it derives from the fact that the list of the output values contains all the possible values exactly once.

3.4.2 Low-degree distinguishers

The fact that a function has low degree may be used for both attacks and distinguishers, depending on the setting. A notable example is the second preimage attack on Hamsi [DS11b]. The idea of the Hamsi attack is to exploit low-degree polynomials that produce the image bits of the compression function. Not only the degree of these polynomials is low, but also the diffusion of the Hamsi round function is slow. As a result, the attacker is able to evaluate particular output bits of the compression function on all 2^{32} message words faster than by 2^{32} calls of the compression function. However, the total complexity gain is rather small, and the authors had to enhance the attack by taking the IV into account.

High-order differentials are also used as *zero-sum* distinguishers for a permutation [AM09]. Indeed, assume that both the input and the output of a permutation are low-degree polynomials of an intermediate state S:

$$I = f_1(S),\ O = f_2(S),$$

where S can be a single word or a set of boolean variables (with some of them fixed to constant). If

$$\max\ (\deg\ f_1,\ \deg\ f_2) < k,$$

then one constructs 2^k inputs that sum to zero and their outputs sum to zero as well. It is easy to prove that this problem for a random n-bit permutation can not be solved in less than in $2^{\frac{n}{2^k}}$ calls. Therefore, if k is small enough, this property demonstrates non-ideal behavior of the permutation. An algorithm for the generalized birthday problem by Wagner [Wag02] provides a significantly higher upper bound ($2^{\frac{n}{1+k}}$). Note that degree estimates can be improved if a primitive has an SPN design with S-boxes of small degree [WHYK10, BCC11, DL11].

If the adversary restricts to a particular large k, she gets a weaker scenario. However, the high-order method may now outperform the generalized birthday algorithm and similar attacks [BM97]. Note that with k close to n the partition of the full domain into zero-sums may be described efficiently [BCC11].

3.4.3 Cube attacks

Cube attacks [DS09] can be viewed as an extension to high-order differentials. Now we sum over different inputs in order to get a linear function of secret parameters. Let us consider a function $f_K(\overline{x})$ of a vector-variable $\overline{x}$ and a parameter K. Consider also an index set $I = \{i_1, i_2, \ldots, i_k\} \subseteq \{1, \ldots, n\}$ and factorize f_K by $X_I = x_{i_1} x_{i_2} \cdots x_{i_k}$:

$$f_K(\overline{x}) = X_I \cdot p_K(\overline{x}) + q_K(\overline{x}).$$

If no monomial in q_K contains X_I, the sum of f_K over cube I is equal to p_K, which is called a *superpoly*:

$$\bigoplus_I f_K(\overline{x}) = p_K(\overline{x}).$$

If p_K has degree one as a function of K, i.e. it is linear, the monomial X_I is called a *maxterm*. Given sufficiently many linear relations, we solve the linear system and find K.

The main challenge in cube attacks is the search for an appropriate maxterm, which is typically heuristic. First, the complexity of the attack grows exponentially with the size of I, so maxterms should be as short as possible. The main principle of the maxterm search is to fix I randomly, and then add indices to it unless the superpoly becomes linear. If the superpoly is reduced to a constant, we restart the search.

Another problem is that few block ciphers and hash functions have elegant algebraic representation (MD6 and Keccak are exceptions). As a result, a cryptanalyst has to consider the primitive as a black-box. He tries different maxterms and apply linearity tests in order to detect linear superpolys. The procedure was called a *cube tester* [ADMS09]. The cryptanalyst has to check each candidate maxterm before the attack, which clearly limits all the attacks to those having practical complexity. Various heuristics such as greedy algorithms and other optimization tools are well applied to cube testers [Sta10].

Dynamic cubes. The dynamic cube attack [DS11a] is an extension to the basic attack with the following improvements. First, we work with a more general expression of f_K:

$$f_K(\overline{x}) = P_1 P_2 + P_3,$$

where P_i are (almost) arbitrary polynomials of K and $\overline{x}$. The second improvement is that we guess the key bits and adapt the cubes to sum over for each guess. During the attack we try all possible guesses and sums to nullify P_2 and thus work with reduced P_3. This method is by all means ad-hoc and less generic, and has higher requirements to the algebraic representation of a primitive.

Another problem is that due to the key guess the attack loses its deterministic behavior. As a result, a cryptanalyst puts significant effort to figure out the success rate and the time complexity of the attack. The latter parameter, in contrast to traditional probabilistic attacks, depends greatly on the algebraic representation. Even worse, the details that lead to the attack may also lead to the increase in time complexity as they are explicitly non-random. It is unclear yet whether the parameters of the recent attacks are estimated correctly [DGP+11].

3.4.4 Other attacks

The key recovery and collision search problems can be expressed in terms of solving systems of nonlinear equations. Therefore, generic methods for nonlinear system solving, like Gröbner basis methods [CLO07] and their weaker adaptations XL and XSL [CKPS00, aJP02], can be applied. However, the complexity of these algorithms appears to be exponential for generic systems, and there is little evidence how to convert any such attack to practica

3.5 Data-dependent operations

The data-dependent operations are one of the most controversial design concepts. We say that an operation is data-dependent, if it is expressed as a family of functions indexed by input data. The examples are key-dependent S-boxes, key- and message-dependent rotations and permutations. The advantage of data-dependent operations is that its statistical and diffusion properties are largely unknown to the attacker, so that she can not predict the behaviour of the primitive on a large set of data.

On the other hand, the advantage quite often converts to the disadvantage. Indeed, if the statistical property is key-dependent, then by statistical parameters of a sample yield information about the key. The main difficulty of the analysis is hence the extraction of those properties and relating them to a particular key group.

Another problem is that some keys may be weak since they produce weak operations. The cipher IDEA injects short subkeys by both multiplication and addition. The early analysis demonstrated that subkeys 1 and -1 eliminate the effect of multiplication and hence make the full cipher perfectly linear [DGV93].

An SPN design PrintCipher precedes each S-box with a key-dependent permutation. The S-box difference distribution table has several entries $\alpha \rightarrow \alpha, \mathrm{wt}(\alpha) = 1$, among those with the highest probability. As a result, the best differential trail works with one-bit differences, whose positions are determined by the key-dependent permutation. Then the adversary simply encrypts pairs of plaintexts differing in one bit and check if the ciphertext pairs also differ in one bit. The right pair provides information on the actual trail and hence on the key [ALZ11]. This approach can be generalized to ciphers, whose S-boxes are key-dependent and have weak differential properties [BKLT11].

The primitive ARMADILLO is based on input-dependent permutations. In several settings an adversary has control over a part of the input, and hence know the permutation parameters up to a large extent. Furthermore, the adversary can manipulate the diffusion in the internal state by controlling the permutation parameters, and hence perform meet-in-the-middle and other types of attack rather easy [ABNP$^+$11].

The hash function DynamicSHA family employs message-dependent word rotations. An adversary finds the message values that yield the best differential trails in terms of probability and collision construction [ADIP09].

Chapter 4

Attacks on byte- and word-oriented primitives

We distinguish bit-oriented primitives (the most famous example is DES) from byte- and word-oriented primitives, which operate rather on bytes and words than on bits. The main difference is the existence of bytewise operations, e.g. S-box layers, and the abscence of bit-oriented transformations such as bit transpositions.

The main property exploited in byte-oriented attacks is that bijective bytewise operations preserve *variation in a single byte*. The variation is a non-zero difference in the differential cryptanalysis and a multiset in multiset attacks. An attacker follows the variation through a primitive easily, because the diffusion is provided only by inter-byte operations, such as mixing layers. As a result, both the analysis and the design are simplified.

Contents

4.1 Multiset attacks

4.1.1 Definitions

The first multiset attack was named the *square attack* [DKR97], because it was applied to the block cipher SQUARE. The idea is quite simple: consider a set Λ of internal states such that they differ in only one byte i_0 and this byte takes all possible values, i.e. there are 2^8 states. Formally,

$$\forall (x,y) \in \Lambda \quad \begin{cases} x_{i_0} \neq y_{i_0}; \\ x_i = y_i \; \forall i \neq i_0. \end{cases}$$

If a bytewise bijective transformation (e.g., a layer of S-boxes) is applied to Λ, the property still holds. Diffusion makes other bytes active, though these bytes might not take all possible values. Two rounds of SQUARE (and AES) make all the bytes active, and each byte takes all possible values. Now we show to get a quite powerful distinguisher from this fact.

We provide the notation by Biryukov and Shamir [BS01] as the most complete. A *multiset* of m-bit values is an unordered tuple, where values can repeat. Consider, for example, 1000 ciphertexts, and the first byte in each ciphertext. Then the 1000 corresponding values form a multiset. A set of k-byte internal states, produced of different inputs, is a *block* of 8-bit multisets: some bytes may get the same values.

A multiset has *property C (constant)* if it consists of repetitions of the same value. A multiset has *property E (even)* if each value occurs an even (or zero) number of times. Properties C and E are preserved by arbitrary functions.

A multiset of m-bit values has *property P (permutation)* if it contains exactly once each of the 2^m possible values. Property P is preserved by arbitrary bijective function.

A multiset has *property B (balanced)* if the XOR of all values is zero. Any multiset with property E or P also has property B.

A multiset has property D (*dual*) if it has property P or property E.

4.1.2 Properties

The properties of multisets can be illustrated on SASAS [BS01], a scheme that alternates layers of S-boxes with affine transformations, and AES (Figure 4.1).

Denote by S^k a block of k word multisets with property S. Consider a multiset PC^{k-1} where each multiset enters a bijective S-box. The multiset is preserved by the first S-box layer (see properties in Section 4.1.1).

Let us prove that the next layer, which is an affine transformation, keeps the weaker property D^k, i.e. permutation or even in all the words. Indeed, a word W is a linear function of k variables. All but one of these variables are constant: $W = Lx + B$ where x is the word with property P. For each W the equation either has no solution, or 2^t solutions, i.e. the word has property D. In the case of AES the linear transformation of the first round converts PC^{15} to P^4C^{12}, then it is preserved by S-boxes, and then transformed to P^{16} by the linear transformation of the next round.

Notice that the second layer of S-boxes in SASAS preserves D^k. However, we have to weaken the property so that it is preserved by an affine transformation. Since D implies ($\rightsquigarrow$) B, we now consider a multiset B^k. The third S-box layer in AES preserves the multiset P^{16}.

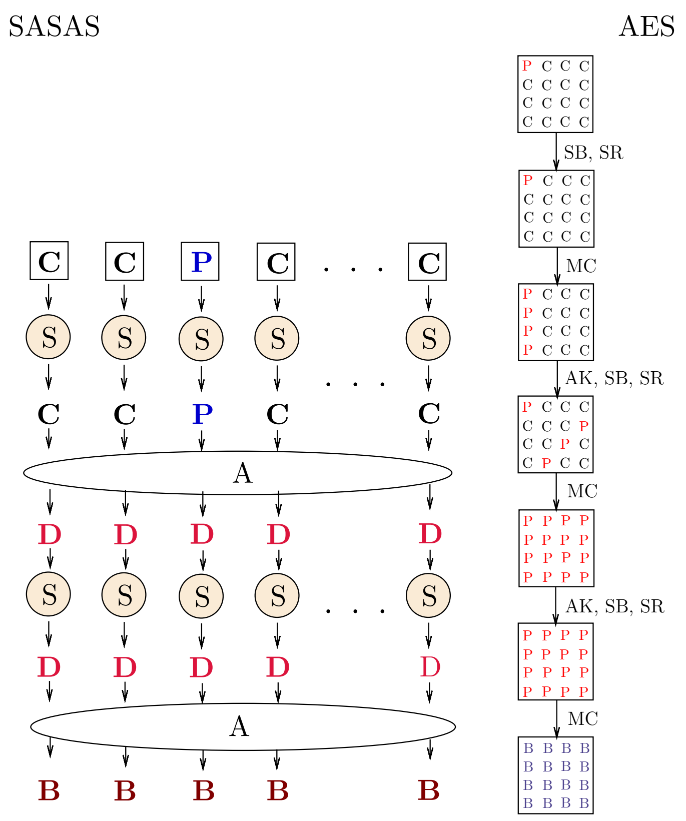
The second A-transformation in SASAS preserves B^k since the B-condition is linear. No simple property is preserved by the last S-box layer, but it is enough to construct a distinguisher. For AES the third linear transformation makes all bytes balanced (B), and the resulting 3-round pattern is called an *integral* [KW02]:

$$\begin{aligned}
\text{SASAS}: &\quad PC^{k-1} \xrightarrow{S} PC^{k-1} \rightsquigarrow D^k \xrightarrow{A} D^k \xrightarrow{S} D^k \rightsquigarrow B^k \xrightarrow{A} B^k;\\
\text{AES}: &\quad PC^{15} \xrightarrow{S} PC^{15} \rightsquigarrow P^4C^{12} \xrightarrow{A} P^4C^{12} \xrightarrow{S} P^4C^{12} \rightsquigarrow P^{16} \xrightarrow{A} P^{16} \xrightarrow{S} P^{16} \rightsquigarrow B^{16} \xrightarrow{A} B^{16}.
\end{aligned}$$

4.1.3 Extensions

High-order integrals. If an integral is a union of integrals, it is called a *high-order integral* [KW02]. The point is that a high-order integral can be longer than its elements. If we extend the AES integral one round back, we get a multiset M, where four bytes take 256 randomly looking values. However, a set of 2^{24} different such multisets form a P^4 multiset in these bytes. As a result, we get a 4-round *high-order integral*, which ends with a B^{16} multiset, now for 2^{32} texts. For versions of Rijndael with larger (160 bits and more) blocks it is possible to construct a 16-round integral [JdFP05].

Applications to non-SPN structures. Multiset attacks can be applied also to Feistel schemes [HQ01] and bit-oriented transformations [ZRHD08]. The principles are exactly the same, no new property is introduced, and the attacks are rather based on slow diffusion of the



SASAS
AES
C C P C . . . C
S S S S . . . S
C C P C . . . C
A
D D D D D
S S S S . . . S
D D D D D
A
B B B B B
P C C C / C C C C / C C C C / C C C C
SB, SR
P C C C / C C C C / C C C C / C C C C
MC
P C C C / P C C C / P C C C / P C C C
AK, SB, SR
P C C C / C C C P / C C P C / C P C C
MC
P P P P / P P P P / P P P P / P P P P
AK, SB, SR
P P P P / P P P P / P P P P / P P P P
MC
B B B B / B B B B / B B B B / B B B B

Figure 4.1: Multisets for SASAS and AES.

primitives, than on the non-trivial property of operations. The fact that an S-box preserves property P still remains the key element of any multiset attack. Behaviour of multisets in Feistel schemes with slow diffusion is also described via permutation polynomials [LSLQ10].

Multisets can be used also in primitives with modular additions and multiplications, like IDEA [DST04, BDK07] and SNOW3G [BPSZ10]. It is easy to prove that $P \boxplus C$ gives P, and similarly $P \odot C$ gives P.

4.1.4 Functional collision

The functional collision method was designed for AES [GM00] and for the long time remained the best attack on 7 rounds of AES-128. The idea of the method is the following. Consider some internal variable x as a function of plaintext P with key K as a parameter:

$$x = f_K(P).$$

Assume that P can be splitted into three groups: one byte y, a group Z, and a group U of the other bytes such that x as a parameterized function of y and Z can be expressed as

$$x = g_{K,U}\left(h_0(y), h_1(Z), h_2(Z), \ldots, h_t(Z)\right),$$

where h_i are different functions, and

$$t < 2 \cdot |Z|. \tag{4.1}$$

Also, h_0 can be a vector-function. Then there exist $Z' \neq Z''$ such that $h_i(Z') = h_i(Z'')$. As a result, last t arguments of g are equal for different Z, so

$$f_K(y, Z'') \equiv f_K(y, Z''),$$

and the two functions, that define x, are equal as well.

This is a very powerful distinguisher, since after such pair (Z', Z'') is found, the property holds for 2^8 values of y.

Applications to AES. The variable x in AES is an arbitrary byte of A^4 (the internal state after 3 rounds and the SubBytes operation) and y is an arbitrary byte of the plaintext. For the sake of simplicity, let both of them be the byte $a_{0,0}$ in the state. Then Z is the other three bytes in the same column of the plaintext: $a_{1,0}, a_{2,0}, a_{3,0}$. Here Z is chosen so that all the four bytes are shifted to different columns and affect each other only in the second round by the linear transformation.

As a result, any diagonal byte $b_{i,i}$ in the input of the third round is defined as follows:

$$b_{i,i} = s_i(y) \oplus h_i(Z),$$

and

$$x = g_K(b_{0,0}, b_{1,1}, b_{2,2}, b_{3,3}).$$

Now check the condition of Equation 4.1. The diagonal bytes $b_{i,i}$ are formed of $t = 32$ bits, and the length of Z is 24. Thererfore, there exist two different Z' and Z'' such that $b_{i,i}$ collide as a function of y:

$$s_i(y) \oplus h_i(Z) \equiv s_i(y) \oplus h_i(Z'').$$

Due to the birthday paradox, such a pair exist among any 2^{16} different Z. Since y is also a function of four input bytes of the fourth round, this gives a 4-round property. It was used for the 7-round attack [GM00], and later for a 8-round attack on a wider-block version [GM08].

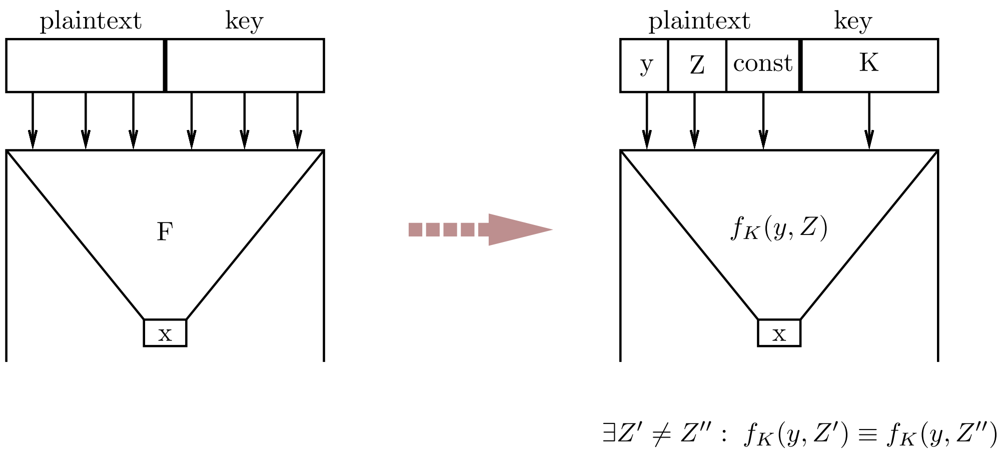


Figure 4.2: Outline of the functional collision attack.

Regroupment of inputs. The variable x can be also expressed as a function of one variable y, which is parameterized by a set of constants, derived from the key and the plaintext:

$$x = f_{C_1, C_2, \ldots, C_k}(y).$$

The number k of constants can be decreased by a careful rewriting of the formula. Gilbert and Minier [GM00] proved that 9 constants (some of them are fixed by U) determine an input byte to the third round. A simple attack first guesses the constants (factor 2^{3k} to the complexity), evaluates x, partially decrypts the last rounds, and check whether the decryption results match the evaluation. To increase the number of attacked rounds some more key bytes must be guessed. The main problem with this method is that the number of constants grows very fast and quickly exceeds the key size, which is the complexity limit.

A 4-round property initially required 25 constants [DS08], and was applicable to AES-256 only. A better idea is to use an (intentionally disordered) multiset and compute the difference between functions of different plaintexts. As a result, a new property requires fewer constants and apparently fits into a precomputed table of size 2^{184}, which leads to an attack on AES-192 [DKS10a]. Another improvement has been proposed in [WLH10].

The paper [DKS10a] also carries a nice explanation of the set of constants and how they are derived. We denote by x_j^i the value of the byte $a_{0,0}$ in the AES state X_j^i in the beginning of round j, $0 \leq j \leq 9$, in the i-th encryption, $0 \leq i \leq 255$, where the plaintexts differ only in $a_{0,0}$. Then the multiset $\{x_j^i \oplus x_j^0\}_{1 \leq i \leq 255}$ is fully determined by

- the full state X_2^0,
- four bytes of X_1^0,
- and four bytes of the subkey added after round 2.

We first prove that the set of all X_2^i determines the multiset, and then prove that this set is determined by the nature of the plaintext structure and the four bytes of X_1^0.

Meet-in-the-middle view. Since the attack is not based on probabilistic events, it may be viewed as a verification of a specific property in the middle of the cipher. Since different key bytes are involved in the partial encryption and decryption, some researchers classify these attacks as meet-in-the-middle attacks [DS08, WLH10].

4.2 Truncated differentials

The main principle of the analysis with truncated differentials is to abstract from concrete values of differences and consider only sets of differences. For example, we may group all the non-zero byte differences into a single set, considering only *active* and *non-active* bytes. The first analysis was made by Knudsen as early as in 1994 [Knu94], though truncated differentials were of limited use while most ciphers were bit-oriented. While diffusion on the bit level may be relatively slow, on the byte level it can be faster, which makes the probabilistic pattern shorter and, thus, less powerful.

The idea is quite simple. Let S be a state of n bytes, and f be a bijective bytewise transformation (e.g., an S-box layer). Consider a pair of states (S, S') with difference only in byte 0. Then $(f(S), f(S'))$ differ in byte 0 and collide in other bytes with probability 1. As a result, the confusion layer does not mix active and non-active bytes. Thus, it is easy to study and exploit the properties of the diffusion layer.

It is natural to use structures (Section 8.4) instead of pairs in the truncated differential analysis. In the previous example a structure with 2^8 states, that differ in byte 0, contains 2^{15} pairs, each passing the S-box layer.

4.2.1 On the edges of characteristics

Truncated differentials are often used on the edges of differential trails, where they replace regular differential trails. Examples include AES differential trails [BK09], which have regular difference in the middle and truncated ones on the edges. Regular differences are truncated in the last rounds, where no restriction is put on the output difference of active S-boxes. In the forward direction this happens deterministically, while the backward transformation $* \to \delta$ holds with probability 2^{-8}. Also, this approach weakens the output filter, so the number of candidate pairs (or quartets) increases.

On the plaintext side we submit a structure such that positions with active S-boxes in the first round get sufficiently many values. The precise formula involves the diffusion parameters of the primitive and is quite complicated (see [BDK02] for the case of the boomerang attack), so we consider a simple case. Let one plaintext byte be active with unknown difference, and let it be converted to a fixed difference δ after the key whitening and an S-box layer. Then we get an equation of type $S(x+k) = S(x'+k) + \delta$, where k is the key, and (x, x') is the plaintext byte pair. Evidently, there are 2^8 (out of 2^{16}) pairs satisfying the equation. A structure with this active byte would have 2^8 plaintexts, so 2^8 right pairs come out of the first round.

Summarizing, the truncated differentials help to keep the probability over the 2^{-n} barrier, but increase the data complexity and the number of candidate pairs for the filtering step.

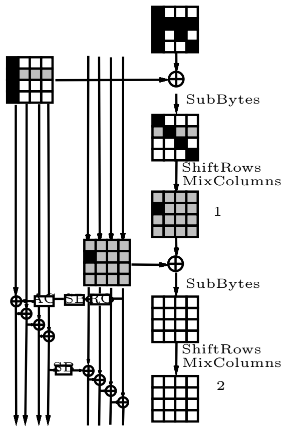


Figure 4.3: Truncated differentials in the 2-round AES trail. Black cells are truncated (arbitrary) differences.

4.2.2 Throughout the primitive

If the diffusion on the word level is relatively slow, truncated differentials may be traced throughout the primitive. A good example is a truncated AES differentialwith the following number of active S-boxes:

$$1 \xrightarrow{\text{Round}} 4 \xrightarrow{\text{Round}} 16.$$

The differential can be preceded by $4 \xrightarrow{\text{Round}}$, resulting in

$$4 \xrightarrow{\text{Round}, P=2^{-24}} 1 \xrightarrow{\text{Round}} 4 \xrightarrow{\text{Round}} 16,$$

which is used in boomerang attacks [Bir04].

The cipher Crypton has weaker diffusion, so as many as 8 rounds were attacked with truncated differentials [MG00].

In general, however, truncated differentials quickly spread to the whole internal state due to the diffusion, and this happens faster than for fixed differentials. It is undesirable in attacks on block ciphers, but is tolerated in the attacks on hash functions, such as the rebound attack 4.3.

Attacks on stream-based hashes

Truncated differentials were successfully applied to word-oriented hash functions, such as RadioGatun [Kho08] and Grindahl [Pey07, Kho09a]. These primitives process relatively small message blocks with a compression function, which is also not assumed to be ideal and, thus, has exploitable differentials. The message block can be chosen arbitrarily, which gives additional freedom (see Section 8.4). A compression function might not provide full diffusion (e.g., in Grindahl one active byte affects only four bytes after one call), so the truncated differentials are often used.

The idea of most attacks is to use message freedom to control the difference inside the round function.

4.2.3 Extensions

Symmetric differences

Several operations were found to be suited for *symmetric differentials*, where differences are either $00\cdots0$ or $11\cdots1$. Intra-word rotations and XORs convert symmteric to symmetric. Also, symmetric differences "shrink" all the words to bits, thus making the exhaustive search for the best trails much more efficient. In contrast to simple truncated differences, getting a zero difference out of symmetric differences is deterministic.

For example, the analysis of RadioGatun, whose internal state has 75 words, with symmetric differences is reduced to an analysis of a 75-bit primitive, so a round differential can be described as a 75-bit word (+ 3 bits from the message injection). The trail search is now simplified, so the best trail for RadioGatun spans for more than 100 rounds [FP09]. Other examples are attacks on Arirang [GMK+09], where the symmetric difference is traced through the AES S-box, on Panama [RRPV01,DA07], and on Blake, where it is invulnerable to rotations [GM09].

Linear truncated differences

Another extension is the use of differences that form a linear space $R \subset F_2^n$. The space of differences is larger compared to the symmetric differentials, but the aero-difference is still produced deterministically. In [Kho08] the dimension of the difference space is chosen to optimize the complexity of the meet-in-the-middle attack. The point is that a large dimension increases the probability of a differential, while a small dimension increases the probability of the meet-in-the-middle event.

4.3 Rebound attack

The main idea of the rebound attack [MRST09] is to find solutions for the most expensive part of a truncated differential trail. These solutions may be used for constructing collisions, near-collisions, or any other certificates of non-randomness.

The outline of the attack is as follows:

- Construct a truncated differential trail;
- Find sufficiently many regular differential trails for the most expensive part of the truncated trail.

- Construct a solution (pairs of iterations) conforming to these regular trails (*inbound phase*).
- Check if it conforms to the remaining part of the truncated trail (*outbound phase*).

We consider the primitives that alternate layers $\mathcal{S}$ of non-linear S-boxes with some linear transformations $\mathcal{A}$ (SPN structure):

$$S_i^{\text{in}} \overset{\mathcal{S}_i}{\to} S_i^{\text{out}} \overset{\mathcal{A}_i}{\to} S_{i+1}^{\text{in}} \overset{\mathcal{S}_{i+1}}{\to} S_{i+1}^{\text{out}} \overset{\mathcal{A}_{i+1}}{\to} S_{i+2}^{\text{in}},$$

where S_i^{in} and S_i^{out} the internal state before and after the i-th layer $\mathcal{S}$, respectively. Let S_0 have the highest number W of active S-boxes in the trail.

The simplest variant of the attack works as follows. We construct many regular trails for $\mathcal{A}_{-1}$:

$$\Delta S_{-1}^{\text{out}} \overset{\mathcal{A}_{-1}}{\to} \Delta S_0^{\text{in}}$$

just by following the truncated-differential trail. Since there are only linear operations in this part, the trails have probability 1. Similar trails are constructed for $\mathcal{A}_0$. Then we look for a pair of trails, such that each active S-box in S_0 gets admissible input and output differences (δ_I, δ_O) (non-zero entries in the difference distribution table). Typical S-boxes have two and more solutions for a random pair (δ_I, δ_O) with probability 1/2, and no solutions with the same probability. Summarizing for all active S-boxes, one of 2^W differentials

$$\Delta S_0^{\text{in}} \overset{\mathcal{S}_0}{\to} \Delta S_0^{\text{out}}$$

has non-zero probability and, thus, produces 2^W solutions (Figure 4.4). Then each solution is propagated in both directions and is checked whether it conforms to the full truncated trail.

A solution for the dense part $\mathcal{A}_{-1} \to \mathcal{S}_0 \to \mathcal{A}_0$ is computed with average complexity, though the total complexity is 2^W at least. As a result, we exclude the active S-boxes in S_0 from the probabilistic phase of the attack.

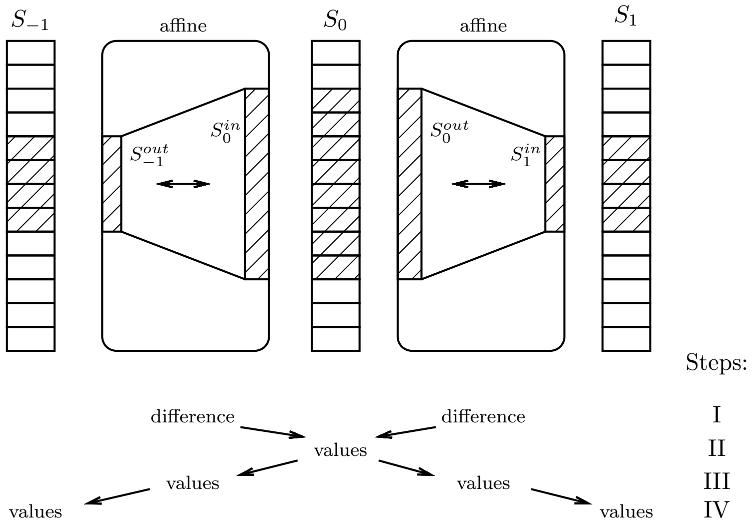


Figure 4.4: Rebound attack.

The attack has many optimizations.

4.3.1 Gradual difference filtering

The differences in the inbound phase can be matched gradually as follows [MPRS09]. Assume that $\mathcal{A}$ is applied separately to q-byte vectors of the internal state (e.g. the AES MixColumns is applied to 4-byte columns). For the sake of simplicity, let this subtransformation convert one active byte in S_{-1}^{out} is converted to q active bytes in S_0^{in}. Then we try 2^8 possible differences for this byte, compute q bytes of ΔS_0^{in} and derive a 1-bit restriction for each corresponding byte of ΔS_0^{out}, which implies a total q-bit restriction on ΔS_1^{in}. As a result, we try only 2^7 assignments for one of ΔS_1^{in} bytes, compute back to ΔS_0^{out}, and get new restrictions on the ΔS_0^{in}, and so on. In the end we construct all possible solutions for $\mathcal{S}_0$.

This approach was generalized and improved by Naya-Plasencia in [NP11], which is an extension of her matching trick in [KNPRS10]. The internal state is splitted into groups of bytes/words so that the differences can be matched and sieved in parallel.

4.3.2 Injection as a source of freedom

Assume that there is an injection from the key or message schedule between $\mathcal{A}_i$ and $\mathcal{S}_{i+1}$:

$$S_i^{\text{in}} \xrightarrow{\mathcal{S}_i} S_i^{\text{out}} \xrightarrow{\mathcal{A}_i} \cdot \xrightarrow{\oplus K} S_{i+1}^{\text{in}},$$

and K can be chosen independently.

Then both layers $\mathcal{S}_0$ and $\mathcal{S}_1$ may have the maximal number of active S-boxes, and we construct solutions as follows. First, we find regular differential trails for $\mathcal{A}_{-1}$ and $\mathcal{A}_1$. Then we assign a random value to the internal state S_1^{out}, and compute the difference backwards through $\mathcal{S}_1$ up to $\mathcal{S}_0$. Due to the injection value, which is not fixed yet, S_0^{out} is unknown. However, each active word in $\mathcal{S}_0$ gets input and output differences. We find a solution if it exists, and connect it to $\mathcal{S}_1$ by fixing an appropriate value of K.

This idea can be extended to more layers, if there are several injections that can be chosen independently.

4.3.3 Multiple inbound phases

The inbound phase can be mounted in different parts of the truncated trail independently if the S-box layers in these phases are not fully active. Starting from the solutions for $\mathcal{S}_i$ and $\mathcal{S}_j$, we meet in an intermediate state and filter out incompatible pairs. The remaining solutions are computed through the remaining part of the truncated trail (Figure 4.5).

Formally, assume that W bytes are active in both layers $\mathcal{S}_i$ and $\mathcal{S}_j$, and 2^W solutions are constructed in each inbound phase. The solutions are computed to a middle point P where they match in q bits. Since a solution is a pair of states, the matching condition is a $2q$-bit condition. Then 2^{2W-2q} valid solutions remain after the meet-in-the-middle .

In the rebound attack on the full Lane [MNPN+09], two layers of the differential trail have 16 (out of 32) active S-boxes. Then 2^{88} solutions are constructed for $\mathcal{S}_i$, and 2^{96} solutions for $\mathcal{S}_j$. The solutions match between $\mathcal{S}_i$ and $\mathcal{S}_j$ in 64 bits. As a result, $2^{88+96-128} = 2^{56}$ solutions are constructed for $\mathcal{S}_i \to \cdots \to \mathcal{S}_j$.

The non-active bytes of the trail can be used as additional source of freedom. If they are not affected by both inbound phases to be merged, the number of solutions just increases. Otherwise these bytes are used to construct a solution for one of inbound phases [SLW+10].

Finally, the internal state may be so large that several inbound phases run in parallel. If the merging conditions are linear (which is typically the case), we get an equation of the following form:

$$S_1 \oplus S_2 \oplus \cdots \oplus S_k,$$

where S_i are the solutions in values to different inbound phases. A solution to the equation may be produced by the generalized birthday algorithm [Wag02], as done in the attack on ECHO [Sch10].

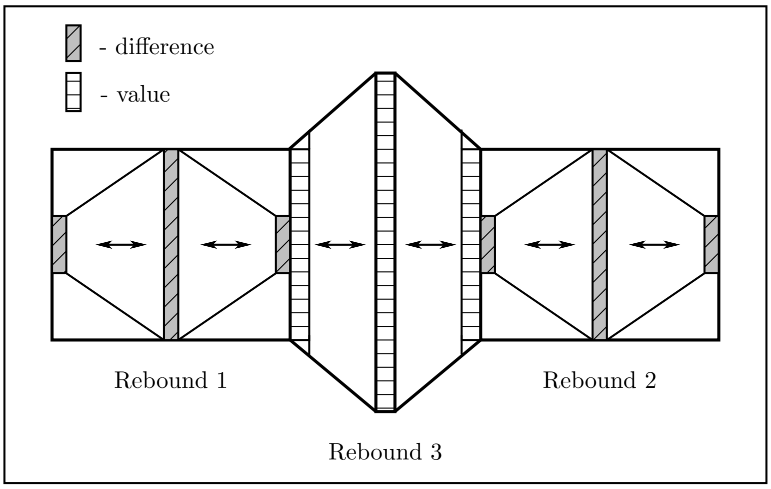


Figure 4.5: Merging solutions in the rebound attack. We find solutions in the inbound phases 1 and 2 and match them in the phase 3.

4.3.4 Extending the inbound phase

One may also construct all the admissible differential paths that span over several $\mathcal{S}$-layers, and only afterwards construct a solution in values. This approach is based on the following assumption that holds for most of S-boxes.

Assumption 1 *If the probability of the n-bit S-box non-zero differential $\delta_I \to \delta_O$ is 2^{r-n}, the right pairs for the differential are affine functions of an r-bit variable:*

$$\exists A, C, b, d: \ \forall x, y: \ (x, x \oplus \delta_I) \xrightarrow{S} (y, y \oplus \delta_O) \qquad \exists u \in F_2^r: \ x = Au \oplus b, \ y = Cu \oplus d,$$

where A and C are matrices, and b and d are vectors.

This holds for any S-box, if $r = 1$, and for the AES S-box, if $r = 2$.

Clearly, the observation holds for a group of active S-boxes, if it holds for each of them. However, it does not hold for non-active S-boxes. Assume that $\mathcal{S}_0$ consists only of active S-boxes. Then any right pair for the differential trail the following system holds:

$$\begin{cases} S_0^{\text{in}} = C \cdot X + D; \\ S_0^{\text{out}} = C' \cdot X + D', \end{cases}$$

where C, C' are matrices, and X, D, D' are vectors. Due to the same principle, the states S_{-1}^{out} and S_1^{in} belong to affine spaces (denote them respectively by $\mathcal{Q}$ and $\mathcal{Q}'$) as right pairs (solutions) to $\mathcal{S}_{-1}$ and $\mathcal{S}_1$ layer differentials. The values for non-active S-boxes in these layers are described as 8-bit variables (there is no need to have a single variable for both input and output values in these layers). Since the layers are connected with a linear function $\mathcal{A}$, we get two matrix conditions:

$$\begin{cases} \mathcal{A}_1(C \cdot X + D) \in \mathcal{Q}; \\ (\mathcal{A}_0)^{-1}\,(C' \cdot X + D') \in \mathcal{Q}', \end{cases}$$

which is a system of linear equations. Its solution is a solution of the three-layer section of the trail with probability 1.

It is also possible to exploit slow diffusion in the message schedule, and thus add one more layer to the inbound phase. In the attack on the full Whirlpool [LMR+09] the similarity between internal rounds and schedule rounds was exploited, so that the solution was constructed for four layers: S_0, S_1, S_2 and S_3.

In the first step we produce 2^{64} solutions for both S_0 and S_3. Then for each pair of solutions (G_0, G_3) we construct an equation of form $f(G_0, G_3, M_0, M_1, M_2)$, where M_i are message blocks injected between S_0 and S_3. Then the authors prove that the equation can be divided into 8 independent equations, which can be solved separately. Together, the solutions define M_i. Note that this approach would fail if the diffusion in the message schedule were better.

Super S-boxes. Another idea is to exploit non-ideal diffusion by grouping parts of the internal state into so-called Super S-boxes [DLP+09]. Super S-boxes are the smallest blocks, which are not mixed with each other during one round. For example, an AES Super S-box is a column in the MixColumn transformation. The rebound attack treats the round as a set of Super S-boxes, thus covering one more round in the inbound phase [GP10, LMR+09]. However, this approach requires more memory for precomputed Super S-box difference tables.

Super S-boxes also provide several levels of difference granularity. The first attacks used fully active Super S-boxes, while the recent ones also consider Super S-boxes, whose inputs are partially active. It may reduce the number of active S-boxes out of the Super S-box layer and reduce the time and memory complexity of the attack [SLW+10].

Neutral bits and message modification. The inbound phase can be extended if the attacked part of a primitive contains neutral bits or auxiliary paths (Section 8.5) that do not destroy a solution to the inbound phase. An example the block cipher Threefish has a related-key differential that has zero difference in the state in eight consecutive rounds. In the rotational rebound attack [KNR10] a solution to the basic inbound phase was constructed in these eight rounds, and then the sufficient conditions for the rotational trail were fulfilled in 5 outer round with the auxiliary path based on the related-key differential.

The attack on Cheetah [WFW09] is an example of the modification on the byte level. Due to a large size of the message block compared to the state size, the change of a single byte of the message may affect the trail up to 4 rounds in both directions.

4.3.5 Trail form and attack layout

A differential trail for a rebound attack is typically constructed ad-hoc. It should be as long as possible, and concentrate active words in as few layers as possible. In attacks on Grøstl [MRST09, GP10, MPRS09], Whirlpool [MRST09, LMR+09], and Luffa [KNPRS10] a trail is based on the expansion of a one-byte difference (e.g., 1-8-64-8-1 in Whirlpool). In attacks on Cheetah [WFW09] and Lane [MNPN+09] the trail is more dedicated.

The index of round where the attack starts is also an important parameter. If the round functions are different (even slightly), the change of the starting position may affect the attack complexity (and even feasibility) significantly. In the attack on Threefish the inbound phase was located to minimize the freedom that is supposed to spend in the outbound phase [KNR10].

Whether chaining value can be taken into account (and hence if the attack translates to the hash function setting) depends on the mode the compression function employs. The MMO mode in Skein admits easy translation. In general, the inbound phase must cover the chaining value, as it is demonstrated in the attack on Grøstl [MRST10].

4.3.6 Rebound in Feistel and ARX schemes

The Substitution-Permutation networks, like AES and Grostl, are natural applications of the rebound attack due to the simplicity of the diffusion, natural bounds on the trail length, and S-boxes with good differential properties.

The ARX schemes are much less suitable for the rebound attack. The first reason is the diversity and abundance of differential trails. As bit-oriented primitives, the ARX systems do not have natural truncated differentials (except those based on single words, which lacks in granularity). As a result, the rebound attacks do not provide the highest number of rounds, and old-fashioned differential attacks are typically better. Notable exceptions are the applications of the rebound attack to a property that is not a differential pattern, and the use of inbound phase techniques to increase the number of rounds. In the attack on Threefish [KNR10] the rotational pair of states is found with small amortized cost in the 15-round inbound phase of the attack and then probabilistically checked for the rotational property in the remaining rounds.

The Feistel schemes are apparently much more vulnerable to the rebound attack than their SPN counterparts. The reason is probably the existence of short local collision trails accompanied with relatively slow diffusion. As a result, an attacker can mount a multiple inbound phase and efficiently use his degrees of freedom. For example, a single inbound phase in the n-bit Feistel scheme may consume as little as $n/2$ degrees of freedom, since the round function makes use of only $n/2$ bits of the state. Two inbound phases which consume the whole n-bit freedom may cover up to 5 rounds in the generic case ([SY11], Figure 4.6), and probably more in concrete compression functions. The Feistel-based compression function of SHAvite-256 has been recently attacked in the rebound style despite a heavy round function with 3 AES rounds [MNPP11].

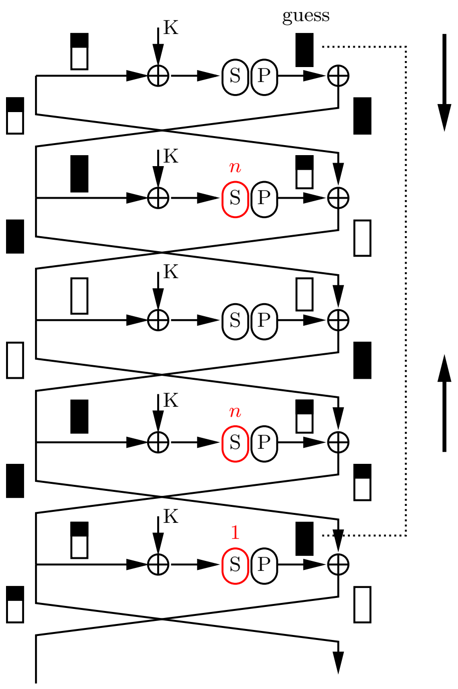


Figure 4.6: 5-round inbound phase in a generic Feistel scheme with SP-round function. $2n + 1$ S-boxes are active.

Chapter 5

Attacks on schedule and injection

The size of the internal state of symmetric primitives is often smaller than the total size of input, because a smaller design is faster and needs fewer operations for good diffusion and confusion. However, this implies that either inputs are *compressed* in the very beginning to match the state size, or portions of inputs are *injected* during the iteration. The injection procedure, which determines the injection values, is called the *schedule* (of the key or of the message). The schedule can be viewed as an additional, separate computation, which operates on the *schedule state*.

In this chapter we consider attacks that exploit weaknesses in the schedule procedure.

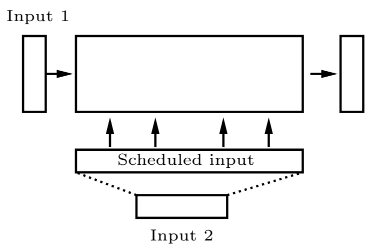


Figure 5.1: Primitive with schedule.

Contents

5.1 Meet-in-the-middle

The meet-in-the-middle (MITM) technique is used to find a secret (or unknown) parameter k of an invertible transformation F_k, given input I and output O:

$$O = F_k(I).$$

For example, in the key recovery attacks F_k is a cipher, k is a key, I is a plaintext, and O is a ciphertext. In the preimage attacks on hash functions, F is a hash function, k is a message, I is an initial value, and O is a hash.

The meet-in-the-middle approach can be applied, if F can be decomposed into independent parts H and G, such that

$$F_k \equiv H_{k_2} \circ G_{k_1}; \quad k = L(k_1, k_2), \tag{5.1}$$

H is invertible, and k_1 and k_2 can be chosen independently.

If G and H have similar computational complexity, we construct $2^{n/2}$ intermediate states $G_{k_1}(I)$ as images of the input, and $2^{n/2}$ intermediate states $H_{k_2}^{-1}(O)$ as preimages of the output. A matching pair

$$Q = G_{k_1}(I) = H_{k_2}^{-1}(O)$$

is found with probability close to $1/2$, so we are able to compute k from k_1 and k_2. The time and memory complexity of the straightforward approach are $2^{n/2}$ permutation calls and internal states, respectively (Figure 5.2).

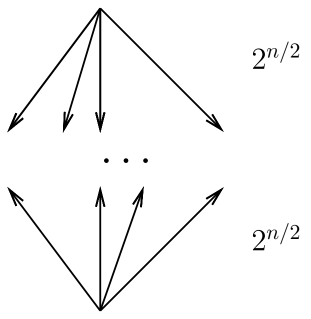


Figure 5.2: Meet-in-the-middle for n-bit functions.

If the full F can not be decomposed as in Equation 5.1, but we are still able to compute an internal variable in the middle:

$$I \xrightarrow{G_{k_1}} q \xleftarrow{H_{k_2}} O,$$

we can detect the mismatch in the middle (*early-abort technique*). In the partial matching method (Section 7.4.2) internal states are partially computed in the last steps, and are matched on a smaller subsets of the state bits. This trick significantly reduces the number of candidate pairs for the MITM. Moreover, the matching bits may be linear functions of "alien" message words as follows:

$$G_{k_1}(I) \oplus L_1(k_2) \overset{?}{=} H_{k_2}^{-1}(O) \oplus L_2(k_1).$$

Then the linear functions are formally moved to the other side of the equation, and we compute them independently (*indirect partial matching*, [ZL10, AGM+09]).

5.1.1 Memoryless MITM

The memory complexity can be significantly reduced with a marginal increase in the time complexity. The most efficient method uses the switching function r that maps elements of the domain to a single bit in a random fashion. Then we define a step function f that evaluates x either to $G(x)$ or to $H^{-1}(x)$, depending on the value of x:

$$f(x) = \begin{cases} G(x) & \text{if } r(x) = 0 \\ H^{-1}(x) & \text{if } r(x) = 1 \end{cases}$$

The function f is iterated by the Floyd cycle finding algorithm (Figure 5.3) as follows. We start from a random value x and use two memory elements $a = f(x)$ and $b = f^2(x)$. In each step we update a by applying f to it, and update b by applying f^2 to it. Upon finding a cycle, we check whether we really have found a pair $G(x) = H^{-1}(y)$ or whether we have found a cycle in G or in H. If the output of r is equidistributed, for each cycle we find $\mathbb{P}(G(x) = H^{-1}(y)) = 0.5$. In case of encountering a cycle in G or H^{-1} we restart the algorithm with another random element x.

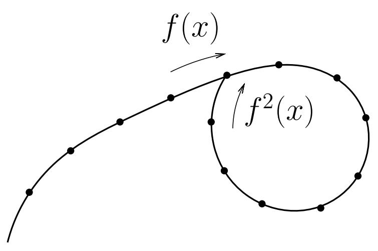


Figure 5.3: Floyd cycle finding algorithm.

5.1.2 Simple key-recovery

The MITM approach is used in key recovery attacks on block ciphers as follows. Assume that an internal variable X depends on only a portion K^F of key bits as a function of plaintext, and depends on only a portion K^B of key bits as a function of ciphertext (Figure 5.4). Given a plaintext-ciphertext pair, the key bits from $K \setminus (K^F \cup K^B)$ are guessed, and X is computed of the plaintext and of the ciphertext. A contradiction implies that the key guess is wrong. This attack benefits from the slow avalanche effect of key bits [BR10].

A 6- and 7-round versions of DES can be attacked, i.e. there exists a good X. It was demonstrated that not only the key bits, but also internal variables can be guessed to speed up the key recovery [DSP07]. Similar attacks on AES cover only 4 rounds [BDD+10].

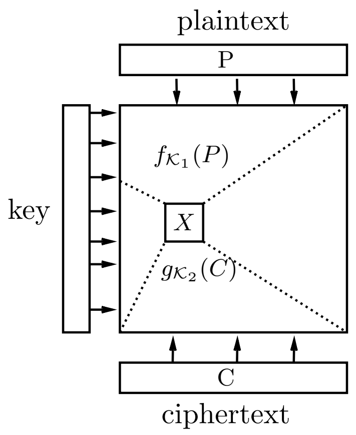


Figure 5.4: Meet-in-the-middle attack on a block cipher.

5.1.3 Other applications

From pseudo-preimage to preimage. The meet-in-the-middle approach is also useful in the preimage search for single call of a compression function. In the latter case only a hash value is fixed. Then the adversary tries to decompose the compression function into two parts, whose inputs can be chosen independently. The complexity of the attack drops according to the number of independent words (see Section 5.2). The attack finds a pseudo-preimage.

A full preimage can be found with another MITM procedure. Assume that a pseudo-preimage can be found in time 2^k, where $k < n$. Then we compute $2^{n/2+k/2}$ images of the original IV: $Q_i = E(IV, M_i)$ with total complexity $2^{n/2+k/2}$. Secondly, we compute $2^{n/2-k/2}$ pseudo-preimages: $E(Q'_i, M'_i) = h$ with complexity $2^{n/2+k/2}$. There exist $2^{n/2+k/2+n/2-k/2} = 2^n$ pairs (Q_i, Q'_j), so we expect to find a matching pair $Q_i = Q'_j$. Then $M_i||M'_i$ is a preimage to h. The total computational complexity is $2^{n/2+k/2}$ There were proposed other methods aimed to reduce memory costs (the computational complexity remains virtually the same): tree methods [Leu08], P3-graphs [CR08], multi-target preimages [GLRW10].

Local collisions. The meet-in-the-middle approach can be used in collision search on the low level, if there exists some independency. For example, in the attack on Tiger [KL06] the zero difference in a variable $z = f(x) + g(y)$ is required, while various differences in f and g can be obtained independently. The meet-in-the-middle technique was applied to get equal differences in them.

Another application to collision search was demonstrated in attacks on GOST [MPR+08] and FORK [Saa07], where the MITM was applied to find actual values of the internal state. In the attack on GOST collisions are found among fixed points that are formed by particular internal variables, which are independent of some message words. Then the fixed points can be found with the MITM. Similarly, in FORK-256 a group of internal variables is summed into a hash, and every variable is independent of some message words.

Difference-based second preimage attacks. In the second-preimage setting the adversary is aware of the full computation that leads to image H. Hence she is able to choose an optimal starting point of the attack, whether it is an internal round of the compression function, or an intermediate call of the compression function. In the difference-based second-preimage attack the adversary works with the differences to the original computations instead of actual values. In the context of meet-in-the-middle attacks, an adversary computes $2^{n/2}$ differences at each direction, and match them in the middle [Kho08, IW09].

5.2 Advanced meet-in-the-middle

5.2.1 Splice-and-cut

In several applications we are less restricted in input and output conditions. I.e., in the pseudo-preimage attack we are allowed to use any chaining value (CV) that matches with the image. Moreover, the CV-image relation can be made a part of the computation. As a result, we do not have to start the meet-in-the-middle attack from the CV or the hash value.

$$H = f(M, CV) = E_M(CV) \oplus CV,$$

where CV is the chaining variable, and E is the block cipher keyed with M. We split the cipher E and message M into three parts: $E = E^3 \circ E^2 \circ E^1$ and $M = M^F||M^B||M^C$ such that

$$M^F \text{ is not used in } E^1 \text{ and } E^3; \qquad \text{and} \qquad M^B \text{ is not used in } E^2.$$

Then we get the following chain of computations:

$$CV \xrightarrow[E^1]{M^B||M^C} S \xrightarrow[E^2]{M^F||M^C} V \xrightarrow[E^3]{M^B||M^C} H,$$

where S and V are the internal states. The basic attack works as follows.

1. Assign arbitrary values to S and M^C.
2. Compute states $\mathcal{V} = \{V \xleftarrow[E^2]{M^F} S\}$ for different M^F and store them in memory.
3. Compute states $\mathcal{V}' = \left\{V \xleftarrow[(E^1 \circ E^3)^{-1}]{M^B} S\right\}$ for different M^B and store them in memory. First, compute CV, then use CV and H to find the output of E^3, and then compute in the backward direction to V.
4. The set $\mathcal{V} \cap \mathcal{V}'$ constitutes preimages for the compression function. If necessary, repeat the procedure for other S and M^C.

The computations in Step 2 and Step 3 are called *forward* and *backward chunks*, and the elements of M^F and M^B are called *neutral bits*. Neutral bits might be bits of round messages and even linear spaces of bits. In the simple case when M^F and M^B have equal length d, the sets $\mathcal{V}$ and $\mathcal{V}'$ have 2^d elements, so the probability of a match is 2^{2d-n}, where n is the state size. Assuming that each chunk costs about 1/2 calls of the compression function, the complexity to find a preimage is about $2^{n-2d+d} = 2^{n-d}$. The complexity grows negligibly, if we match not on the full state, but only on a subset of bits of size about d (Section 7.4.2).

The choice of S_1 and S_2, as well as the decomposition of M, are made ad-hoc so far. Clearly, the longer is the schedule, the harder it is to find such a decomposition.

Another problem is that M^B and M^F may be too small to generate enough states for the meet-in-the-middle. Then we generate new S_1 and M^C and repeat the procedure.

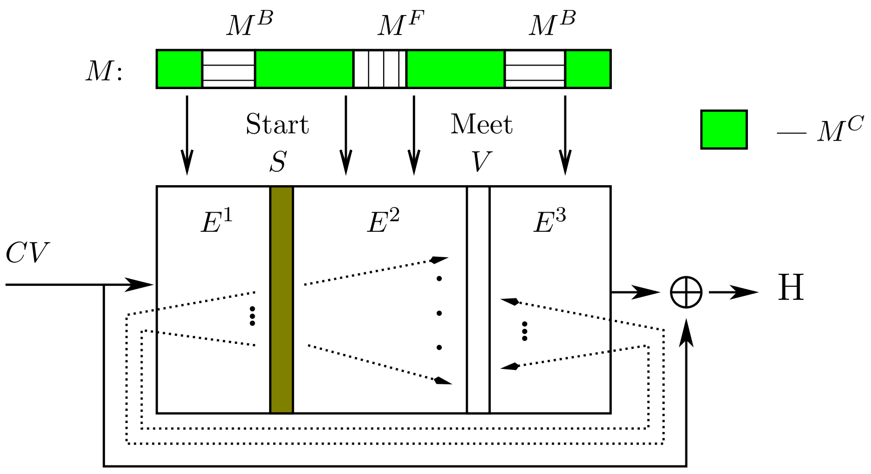


Figure 5.5: Splice and cut.

Double-branch compression functions. RIPEMD is an example of a double-branch compression function: $E(IV, M) = E_1(IV, M) \oplus E_2(IV, M)$. The functions E_1 and E_2 are typically similar, i.e. in RIPEMD they have different round constants. There were proposed two approaches to apply the splice-and-cut procedure [SA09b, WSK+11]:

- Make a starting point in each branch and compute backwards so that the both IVs are indentical. Then the meeting point is the hash value.
- Make a starting point in only one branch. Then compute forward and backwards, compute IV and hash, and then meet in the middle of the second branch.

Key recovery. This technique is also applicable to the key recovery. The wrap-around computation is replaced with the call to the encryption or the decryption oracle. However, there are two important differences.

First, the computation of the plaintext from the ciphertext and vice versa is not bitwise as in the Davies-Meyer mode. As a result, the relevant chunk must cover the full plaintext/ciphertext. In other words, the plaintext/ciphertext must be computed from the starting state without either portion of neutral bits. This condition is difficult to fulfill in the block ciphers like AES, where most of the key material is used every round. The natural application is the ciphers like KTANTAN, where only a small portion of the key is used at each round [WRG+11].

Secondly, the data complexity increases as the starting state goes deep into the cipher. The reason is that the attacker tries all possible values of K^C (former M^C), which affect the plaintext and the ciphertext. If the diffusion is good, this may lead to the data complexity close to the full codebook. The countermeasure is to set the starting state in the first or the last round and to make the starting state as a function of plaintext/ciphertext.

5.2.2 Bicliques

Recently, cryptanalysts have replaced the starting state S with a sophisticated construction called the *initial structure* [AGM+09, GLRW10, AS09]. The broad idea is to put constraints on internal variables and form S out of variables spread over several rounds. However, one has to prove separately that the construction is well-defined, i.e. that a match in the middle implies a correct computation in the start.

The necessary formalism is introduced as follows. First, notice that at Steps 2-3 of the attack a computation of a chunk is always associated with a particular value of neutral bits, either M^B or M^F. Let the backward chunk start at a state Q, and denote by Q_i the state being computed with $M^B = i$. Let also P be the starting state for the forward chunk, and denote by P_j the state being computed with $M^F = j$. If matched at the point V, the combination (Q_i, P_j) must provide a preimage. Therefore, all the Q_i and P_j must follow the following conditions:

$$\forall i,j: \quad Q_i \xrightarrow[\mathcal{B}]{M_i^B || M_j^F} P_j, \tag{5.2}$$

where $\mathcal{B}$ transforms Q to P, and $M_i^B = i,\ M_j^F = j$.

Hence we get 2^{2d} equations linking two groups of states 2^d cardinality each. This construction is called a *biclique* (a complete bipartite graph in the graph theory) of dimension d [KRS11]. A single biclique of dimension d tests 2^{2d} preimage candidates with only 2^d computations of each chunk. Therefore, a pseudo-preimage attack with a biclique of dimension d has complexity about 2^{n-2d} as long as bicliques can be constructed efficiently.

Differential view. The biclique equations (5.2) can be rewritten via a system of differential trails:

$$\nabla Q \xrightarrow[\mathcal{B}]{\nabla M || \Delta M} \Delta P. \tag{5.3}$$

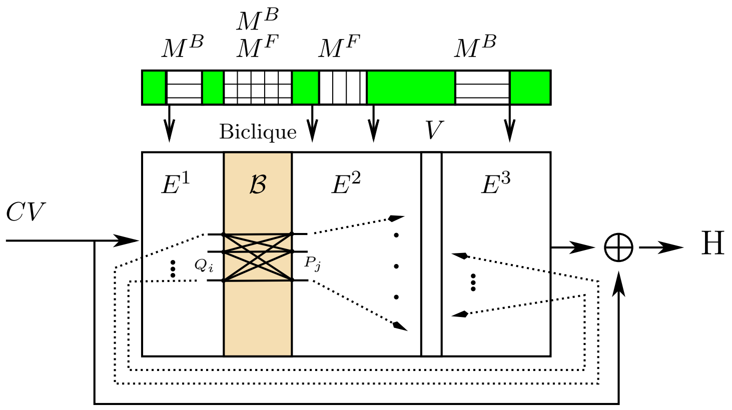


Figure 5.6: Splice-and-cut attack with a biclique

A solution to (5.3) is a solution to (5.2). As a result, a biclique is a solution to the system of differentials, and can be constructed with tools from the differential cryptanalysis. If the differential trails $\nabla Q \xrightarrow[\mathcal{B}]{\nabla M||0} 0$ and $0 \xrightarrow[\mathcal{B}]{0||\Delta M} \Delta P$ do not interfere pairwise, a biclique can be constructed rather simple. Otherwise one has to design specific algorithms that reduce the amortized construction costs [KR11, KRS11].

Key recovery. The key-recovery attack is mounted in a similar fashion. The main difference is that the sub-cipher with a biclique is located either in the first or in the last rounds:

$$E: \quad P \xrightarrow[\mathcal{E}_1]{} V \xrightarrow[\mathcal{E}_2]{} S \xrightarrow[\mathcal{B}]{} C,$$

and there exists an internal variable $v \in V$ that can be computed as follows:

$$P \xrightarrow[\mathcal{E}_1]{K^b \text{ not used}} v \xleftarrow[\mathcal{E}_2]{K^f \text{ not used}} S, \tag{5.4}$$

where K^F and K^B are independent parts of the key material. The rest of the key is denoted by K^C.

Similarly, for each K^C we construct a biclique in $\mathcal{B}$:

$$\forall i, j \quad S_j \xrightarrow[\mathcal{B}_3]{K^F=i,\, K^B=j} C_i.$$

Since the number of produced ciphertexts is proportional to the number of bicliques, we have to avoid covering the full codebook when keys are larger than the ciphertexts (like in AES-256). The idea is to keep only those bicliques whose ciphertexts belong to a particular set of cardinality smaller than the codebook.

Neutral bits and message compensation. In this section we have implicitly assumed that the neutral bits are chosen in the initial message, or in the secret key. If the message/key schedule is a transposition, the decomposition is directly translated into the expanded key/message. If the schedule is more complicated, a good decomposition of the original message may not exist. Hence it is natural to select neutral bits in the expanded message, though it brings other difficulties.

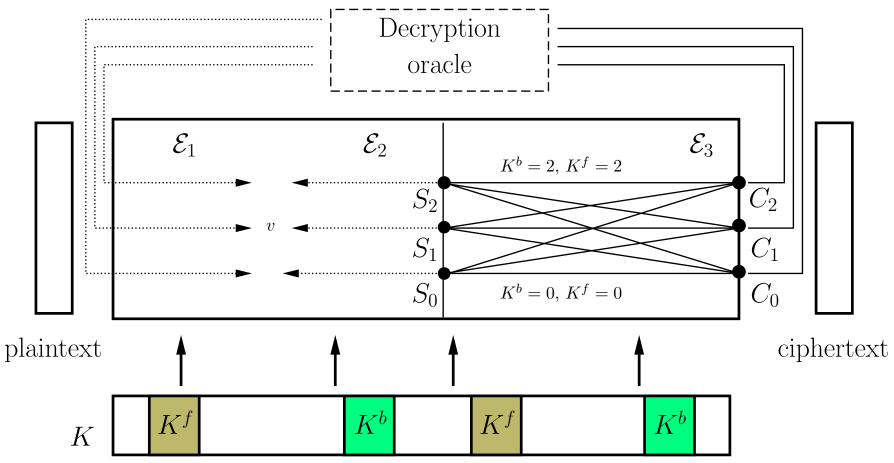


Figure 5.7: Biclique key-recovery attack.

To make it formal, we require the decomposition to be well-defined, i.e. the choice of M^F, M^B, and M^C must uniquely determine the original message. Since the message schedule is typically invertible transformation, it is natural to select M^F, M^B, and M^C in an internal state D of the schedule computation. State D may be merely a concatenation of consecutive message injections. To compute the length of chunks, we investigate the round function MS of the message schedule. We identify the part of state $D_{next} = MS(D)$ that depends on M^B and figure out what are the corresponding message injections. These injections mark the end of the forward chunk. Analogously, we identify the part of state $D_{previous} = MS^{-1}(D)$ that depends on M^F, and compute the length of the backward chunk.

To make the chunks as long as possible, we may select neutral blocks in internal variables of the schedule computation that do not necessarily form a full state. The procedure is called *message compensation* and is exceptionally heuristic [AS09, AGM⁺09, GLRW10, KRS11]. The constants used in those procedures are exactly the message block M^C. The use of bicliques puts an additional restriction: to construct bicliques easier one aims to find sparse differential trails, which means that the corresponding message injections should not depend on neutral bits.

5.3 Local collision

A *local collision* is a differential that starts and ends with the zero difference in the internal state, but is non-zero in the middle. The schedule state may have arbitrary difference, so the non-zero difference appears and disappears as a result of the injection from the schedule. The first injection of the non-zero difference is called a *disturbance*, and the last injection, that cancels the difference, is called a *correction*.

5.3.1 Collision search

One of the first applications of local collisions was the cryptanalysis of SHA-0 [CJ98]. The consecutive injection of differences in particular bits of SHA-0 message blocks results in the zero difference in the state after 6 rounds. Let $\xrightarrow{M}$ denote the round function with a message block M, which operates on internal states I_k. Then the following pattern holds with high

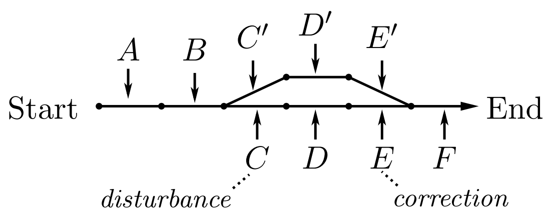


Figure 5.8: Local collision outline. $A, B, \ldots, F$ — inputs.

probability:

$$I_0 \xrightarrow{M_1} I_1 \xrightarrow{M_2} I_2 \xrightarrow{M_3} I_3 \xrightarrow{M_4} I_4 \xrightarrow{M_5} I_5 \xrightarrow{M_6} I_6.$$
$$I_0 \xrightarrow{M_1\oplus\delta_1} I_1' \xrightarrow{M_2\oplus\delta_2} I_2' \xrightarrow{M_3\oplus\delta_3} I_3' \xrightarrow{M_4\oplus\delta_4} I_4' \xrightarrow{M_5\oplus\delta_5} I_5' \xrightarrow{M_6\oplus\delta_6} I_6.$$

All the δ_i are one-bit differences. The probability of this characteristic varies from 1/8 to 1, depending on the round index.

If all the message blocks were independent, one local collision would be enough for a global collision. Due to the message schedule the differences δ_i spread to other round injections. Due to the linearity and the self-similarity of the message schedule, all the δ_i from the first 16 blocks appear in the same positions in the further blocks. As a result, the local collisions in the first rounds are reproduced later, as they were geometrically translated. The resulting differential trail is an overlap of several local collisions, and its probability is the multiplication of local collision probabilities.

Since the last 64 message words of SHA-0 are linear combinations of the first 16 words, the concatenation of all the words is a codeword of a linear code. The difference vector is another codeword, and the number of local collision is related to its weight. More precisely, the number of local collisions is the weight of the codeword produced by the expansion of the disturbance difference δ_1 (or any other δ_i) from the first 16 words.

A collision trail starts and ends with the zero difference, which puts constraints on the positions of local collisions. This is too restrictive, so a multi-block collision, whose trails for compression functions do not end with the zero difference, can be found faster [BCJ+05]. In attacks on SHA-1, DynamicSHA and others [WYY05, Saa07, ADIP09] the local collision approach was significantly improved.

A local collision was also used in the construction of auxiliary differential paths [NSS+06]. The benefit here is a low weight of differences in the trail, which allows to precisely impact bits in the modification.

The same approach can be applied to other hash functions with regular message schedule. If the schedule is a transposition (MD5, Blake), then the local collision approach is inefficient, because the transposition rarely preserve the positions of local collision message differences.

5.3.2 Meet-in-the-middle attacks

Key recovery

A local collision can be used in the meet-in-the-middle attack on block ciphers (Section 5.1.2) as follows. Assume that we attack a block cipher whose key schedule produce short subkeys (compared to the total key length). Assume also for a given plaintext P it is easy to derive pairs of the first subkeys $\{(K_i^0, K_i^1)\}$ that transform P to the same internal state S. Then these

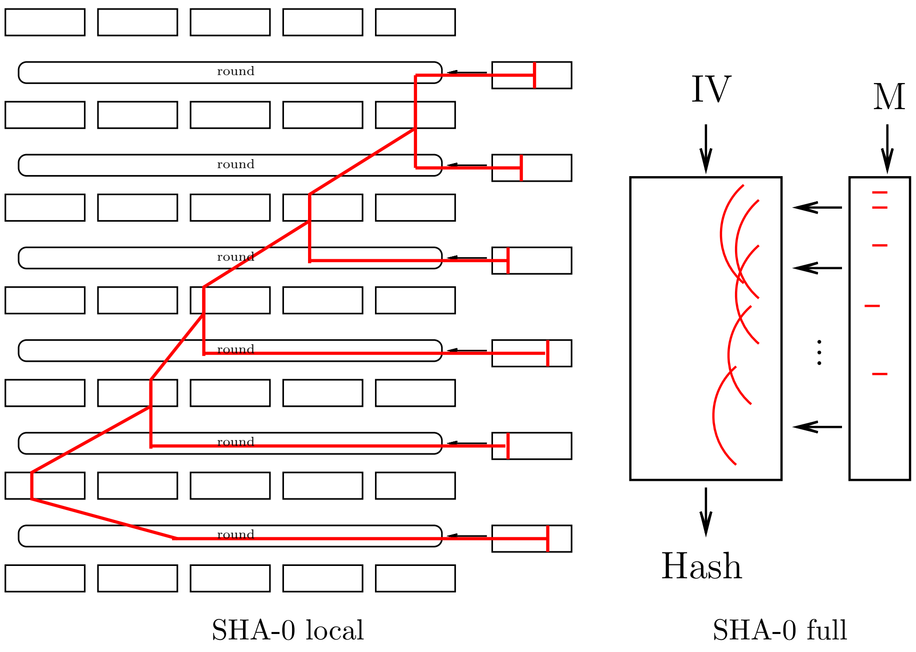


Figure 5.9: Local collision in SHA-0 and a full differential trail.

pairs are actually neutral with respect to the computation of S. As a result, an attacker tries different pairs of subkeys while decrypting the ciphertext and hence gain additional branches in the meet-in-the-middle attack.

In the attack on GOST [Iso11] the 4-tuples of subkeys have been found for both the plaintext and the ciphertext so that a 16-round section of GOST with subkeys

$$(K^0, K^1, \ldots, K^7, K^0, K^1, \ldots, K^7)$$

is attacked in the meet-in-the-middle fashion.

Preimage search

Local collisions are used in the preimage search as an improvement to the splice-and-cut technique (Section 5.2).

The first idea [Leu08, SA08b] is to use the colliding message parts as a degree of freedom for the remote control on the state. Indeed, assume that we fix state S in an attack, and message blocks M and M' form a local collision in a subcipher E that follows S:

$$E_M(S) = E_{M'}(S).$$

Then a switch from M to M' does not change the computation of E, but may provide an easy way to fulfill some conditions after E. This technique was used to control the output of the compression function.

The second idea is to use a local collision as a part of the initial structure (apparently, in the biclique). We fix internal variables in a biclique so that the local collision happens with high probability. As a result, the message differences create the biclique differentials that do not affect each other, which yields a biclique. A probabilistic approach to this idea has been introduced in [SA08a].

In the double-branch compression functions like RIPEMD one is able to construct local collisions that starts in the first round of one branch, goes backwards, then repeats in the second branch and disappears [WSK+11]. Hence two appearances of an "alien" neutral word cancel each other.

5.3.3 In related-key attacks

The key difference in a block cipher resembles the message difference in a hash function. Therefore, it is natural to construct local collisions inside a cipher in order to get a trail with low number of active non-linear operations.

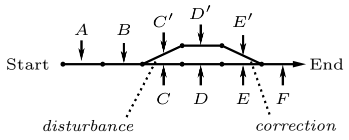


Figure 5.10: A local collision in AES-256.

This approach was used in the cryptanalysis of AES. If the difference is injected in the upper byte, it is not touched by ShiftRows, but is converted to an unknown difference by SubBytes, and then expanded by MixColumns to a full column difference. The full column difference must be corrected by the next subkey addition. The appropriate subkey difference can be predicted with probability up to 2^{-6}, so the local collision trail has probability 2^{-6} (Figure 5.10).

Due to the nonlinearity of the AES key schedule, it is impossible to construct an arbitrary long trail based on local collisions (see also Section 5.4.2).

5.4 Schedule tricks

5.4.1 Schedule recovery

If the intra-word diffusion in the round is slow, one may try to recover a preimage gradually bit-by-bit. Let the message M be expanded to $(M, f(M))$, then then we replace $f(M)$ with M' that consists of independent message blocks. Given the initial constraints, we recover M' and simultaneously minimize the error function $f(M) - M'$ by adjustment of internal variables. For SHA-0 and SHA-1 it was demonstrated that the error can be expressed as a simple formula of internal variables A_i, and the error bits can be gradually fixed to 0 from lower bits to higher ones. Due to rotations in SHA-0/1 this process can not be applied for all the bits, and the remaining error bits are fixed to zero with a dedicated technique. The attack is valid for SHA-0/1 due to the absence of diffusion in the message schedule and small rotation amounts in the round function.

A simplified version of this attack was applied to hash function DynamicSHA [ADIP09]. It can be assumed with reasonably high probability ($\gg 2^{-n}$) that all the data-dependent rotation amounts are zero. This leads to a bit-slice recovery of the internal state, similarly to the attack on SHA-0/1.

5.4.2 Translation through schedule

If the schedule can be defined by a cyclic code or approximated so, some patterns are just copied ("translated") to the further blocks. Let M be an $l = r \cdot k$-bit input to the schedule block (e.g., a 512-bit message in SHA-0), which injects r bits per round. Let the first k injections be just bits of M: $(m_1, m_2, \ldots, m_k)$, and the other t injections be a multiplication of M by a $t \times r$ matrix A, which results in a cyclic code.

Assume that a pattern is generated in rounds i_1–$i_2 < k$. Then it may occur in the further rounds as well, due to the structure of the code. This principle was used in the construction of differential trails for SHA-0/1 [CJ98, WYY05], whose message schedule is given by a cyclic code.

The attacks on AES were constructed in the similar manner. The AES key schedule, though not being a cyclic code, is quite close to it, especially in the backward direction. A local collision for one round of AES-256 was translated to 7 more rounds up [BKN09] (Figure 5.11 demonstrates 4.5 rounds of the key schedule).

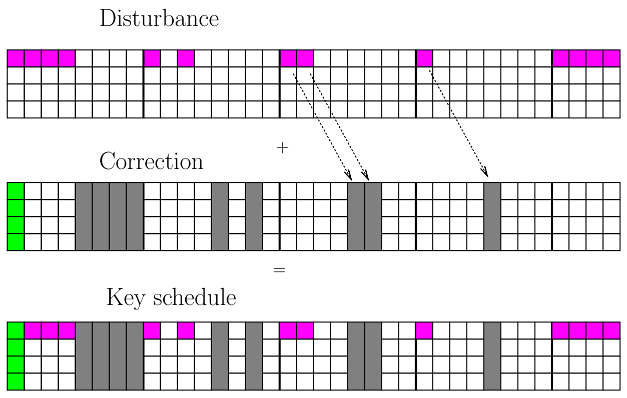


Figure 5.11: Full key schedule difference (4.5 key-schedule rounds) for AES-256.

Chapter 6

Decomposition and combined attacks

The decomposition principle states that a primitive can be divided into several parts, so that optimal attack patterns for each part can be combined into a single attack on the whole primitive. The properties, demonstrated by the patterns, may have different nature. Ideally, the complexity of the resulting attack would be comparable to the product of the complexities of sub-attacks.

The parts of the primitive, where the attacker has some degrees of freedom, are usually treated in a specific way. The examples are the use of structures and truncated differentials around the plaintext and the ciphertext in a block cipher, and the use of ad-hoc characteristics in the first pass of the compression function. Therefore, the most interesting decomposition methods are those working between points of control.

Contents

6.1 Start from the middle

The *start-from-the-middle* is a natural approach when the scenario constraints on the input and output values of the primitive is relatively weak. For example, a collision attack on a compression function imposes constraints only on the hash and IV differences, but not on the values, giving a $2n$-bit condition. In contrast, a collision in the hash function setting imposes also a constraint on the IV value, i.e. a $2n$-bit condition on the input and an n-bit condition on the output, which makes sense to start the collision construction from the beginning thus taking the input condition into account immediately. However, in the compression function setting the attacker may benefit from starting the attack in another point, i.e. in the middle. The *start-from-the-middle* framework splits the primitive into two parts, on which attacks are applied separately.

Evidently, this method can not be applied in the secret-key setting. However, it is quite natural for hash functions, and for block ciphers when comparing to an ideal cipher. The first application of this method was, probably, the collisions for the compression functions of MD4 and MD5 by Dobbertin [Dob96, Dob98]. He solved a system of nonlinear equations, whose solution is a pair of states conforming to the middle part of the differential trail.

Collisions for the compression functions of the SHA family were rarely considered, and most of efforts were put into collision search in hash functions. Collisions for compression functions are motivated by the ongoing SHA-3 competition and various types of attacks on its candidates, e.g. the rebound attack (Section 4.3). In the rebound attack the solutions are constructed for the most expensive part of the differential trail, and then propagated probabilistically through the other parts of the primitive. The complexity of the attack is C/p, where C is the cost of constructing one solution for the middle part, and p is the probability of the trail covering the remaining part of the primitive.

The latter idea is not nessarily used in the rebound framework. The attacker chooses the most optimal method to construct a solution for the middle part, and then apply probabilistic patterns to both input and output. E.g, if the pattern is differential, it leads to differential-multicollision distinguishers [LM11].

The triangulation algorithm constructs a solution for a particular section of a differential trail, which is not necessarily the most expensive one. In the attack on AES-256 [BKN09] a solution is found for rounds 1–5 of AES-256, and in another attack for rounds 3–7.

6.2 Boomerang attacks

The main principle of the *boomerang attack* is to get a *quartet* — four internal states, whose difference form a *rectangle* — in the middle of the cipher, so that the rectangle differences are input differences for trails over a half of the cipher. As a result, one may use short differentials with high probability instead of a long differential with low probability.

The basic boomerang attack [Wag99] is applied to a cipher $E_K(\cdot)$ which is decomposed into $E_1 \circ E_0$. The first sub-cipher E_0 has a differential $\Delta \xrightarrow{p} \Delta'$, and E_1, the second one, has differential $\nabla' \xrightarrow{q} \nabla$, with probabilities p and q, respectively.

We encrypt a pair of plaintexts (P, P') with the difference Δ and apply the difference ∇' to the ciphertexts (C, C') (Figure 6.1). Then a new pair of ciphertexts (D, D') is decrypted. With probability p the first pair has the difference Δ' in the middle: $R = R' \oplus \Delta'$, and with the probability q^2 the pairs (C, D) and (C', D') have the difference ∇' in the middle:

$$R \oplus S = R' \oplus S' = \nabla'.$$

Then $S \oplus S' = \Delta'$, and $Q \oplus Q' = \Delta$ holds with probability p. Finally, we get a *boomerang quartet* (P, P', Q, Q') with probability p^2q^2, while for a random permutation the probability of this event is 2^{-n}.

The boomerang attack works well if there are short differentials with high probability, otherwise the probability p^2q^2 is too low. Another issue is that it is an adaptively-chosen ciphertexts attack, which is a less realistic scenario.

Switch point issues. The set of internal variables, that are the output of E_0 and the input of E_1, are called the *switch point*. This is the point where the quartet rectangle is composed.

First, the differentials may compensate each other. For example, equal differences in some bytes or blocks may result in a higher probability than it is expected (e.g., the S-box switch and the Feistel switch techniques in [BK09]). Moreover, the subciphers can be separated from each other by transformation E' that preserves quartet with reasonable probability. Then we introduce the transition probability r:

$$r = \mathbb{P}\left[(E_0(Q) \oplus E_0(Q') = \Delta') \,\middle|\, (R \oplus R' = \Delta')\&(S \oplus R = \nabla')\&(S \oplus R = \nabla')\right],$$

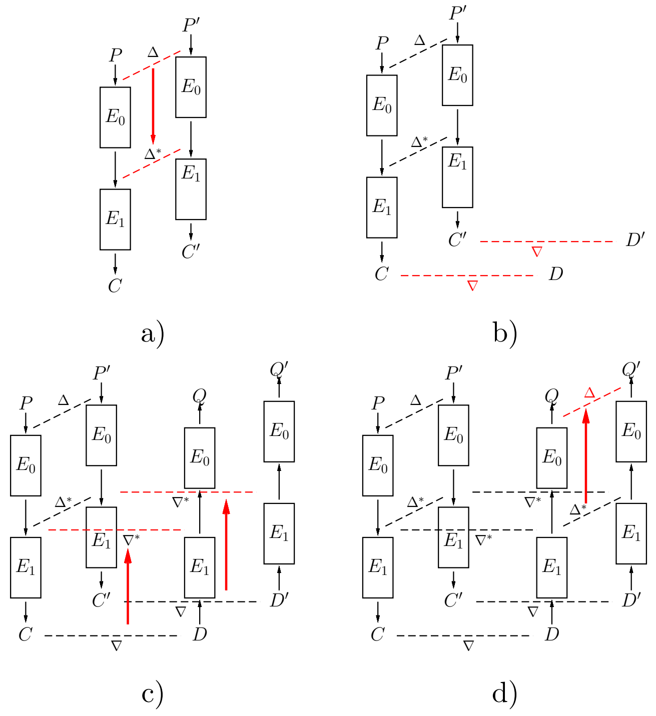


Figure 6.1: Outline of the boomerang attack.

and a boomerang quartet is produced with probability p^2q^2r [DKS10b].

Secondly, the switch point and the differences Δ' and ∇' should be chosen carefully. First, the differentials may contradict each other as follows. Consider a byte variable v at the switch point, such that it has non-zero difference δ_1 in E_0 and non-zero difference δ_2 in E_1. There exist 2^8 quartets of this variable with these differences. Suppose also that δ_2 is an input difference to an S-box differential $\delta_2 \rightarrow \delta_2'$, which is a part of the differential $\nabla' \rightarrow \nabla$. The differential $\delta_2 \rightarrow \delta_2'$ for the AES S-box gives a 6-bit or a 7-bit condition, which is imposed on the both pairs of states in E_1 sides of the boomerang. Clearly, this may filter out all possible quartets (see [Mur09] for concrete examples and Figure 6.2 for an outline). Typically, there is a way to overcome this problem, even gaining in probability, by relaxing the differences at the switching point (see below).

Finally, we note that the choice of variables for the switch point is essential in the correct estimate of the distinguisher probability. The Ladder switch (Section 7.4) is an exploit of the existing parallelism in a design.

Improvements. There have been proposed several improvements of the boomerang attacks. The most significant is the *amplified boomerang attack* [KKS00] (also called rectangle attack [BDK01]), which runs in the chosen-ciphertext scenario. The idea is to encrypt sufficiently many plaintext pairs so that the pairs in the middle with the difference ∇' appear due to the birthday paradox. Due to the quartet property we immediately get the second pair with the difference ∇'. These pairs produce two pairs of ciphertexts with the difference ∇ with total probability q^2. Therefore, we get a quartet. In total, we generate $N^2p^2q^2$ quartets out of N pairs, and the minimal data complexity is $2^{n/2}$. Consequently, the amplified boomerang attack always requires more data and time compared to the original boomerang attack. On the other hand, the amplified boomerang attack admits a truncated difference in the ciphertexts, while

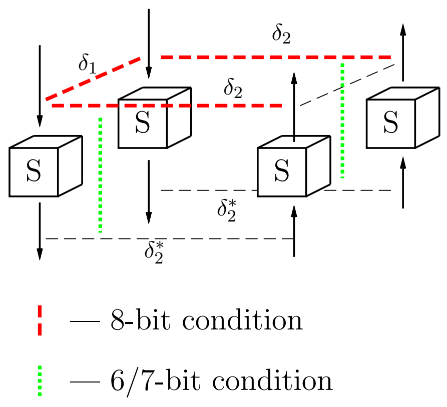


Figure 6.2: Contradiction in the boomerang. The total number of bit conditions for a quartet of bytes exceeds 35.

the regular boomerang attack requires difference to be fully specified ([BDK02] demonstrates how to overcome this problem by partial key guess and, thus, the increase in complexity). This is an advantage if regular differentials for the full cipher have too low probability.

An important improvement was proposed by Wagner [Wag99]: it was noted that the number of good ciphertext quartets is actually higher, since an attacker may consider many rectangles formed by all possible Δ' and ∇' (with the same Δ and ∇). This observation can be applied to both types of boomerang attacks. As a result, the number Q of good quartets is expressed via *amplified probabilities* $\hat{p}$ and $\hat{q}$ as follows:

$$Q = \hat{p}^2\hat{q}^2 2^{-n}N^2,$$

where

$$\hat{p} = \sqrt{\sum_{\Delta'} P[\Delta \to \Delta']^2}; \quad \hat{q} = \sqrt{\sum_{\nabla'} P[\nabla' \to \nabla]^2}. \tag{6.1}$$

The total gain depends on the differential properties of the round function. For example, for the AES S-box with input difference δ we get the following:

$$\hat{p}(\delta) = \sqrt{\sum_{\delta'} P[\delta \to \delta']^2} = \sqrt{2^{-12} + 2^7 \cdot 2^{-14}} \approx 2^{-3.5},$$

while a fixed output difference δ' gives 2^{-6} at maximum.

6.3 Combined attacks

The *combined attack* is a sequential application of two probabilistic properties of different kind. The cipher E is decomposed into $E_1 \circ E_0$, so that E_0 and E_1 are weak as nonlinear transformations (see their analysis in Chapter 3):

$$P \xrightarrow{E_0} Q \xrightarrow{E_1} C.$$

6.3.1 Differential-linear combination

The first combined attack [LH94] used differential and linear analysis. Suppose we have a differential $\alpha \rightarrow \beta$ for E_0 with probability 1, and a linear approximation

$$u \cdot Q \oplus v \cdot K = w \cdot C.$$

for E_1 with probability $1/2 + q$. Note that any plaintext pair of form $(P, P \oplus \alpha)$ conforms to the differential, so we derive that

$$E_0(P) \oplus E_0(P \oplus \alpha) = \beta; \quad \Rightarrow \quad u\,(E_0(P) \oplus E_0(P \oplus \alpha)) = u \cdot \beta.$$

After the encryption by E_1, both ciphertexts conform to the linear approximation with probability $1/2 + 2q^2$ due to the piling-up lemma (Section 3.1). Then we derive

$$w(C_1 \oplus C_2) = u \cdot \beta$$

with probability $1/2 + 2q^2$, and use this approximation as a distinguisher (Figure 6.3). We note that β might be defined only in the bits covered by λ_1.

Differentials with probability $p \neq 1$ are used as follows. We specify not β, but its mask $u \cdot \beta \in \{0, 1\}$. Therefore, we consider a truncated differential $\alpha \rightarrow B$, where $\lambda_1(B) = l_b =$ const, which holds with probability $1/2 + p'$ (for a random function $p' = 0$). Then the approximation $w(C_1 \oplus C_2) = u \cdot \beta$ holds with probability $1/2 + 4p'q^2$ [BDK02].

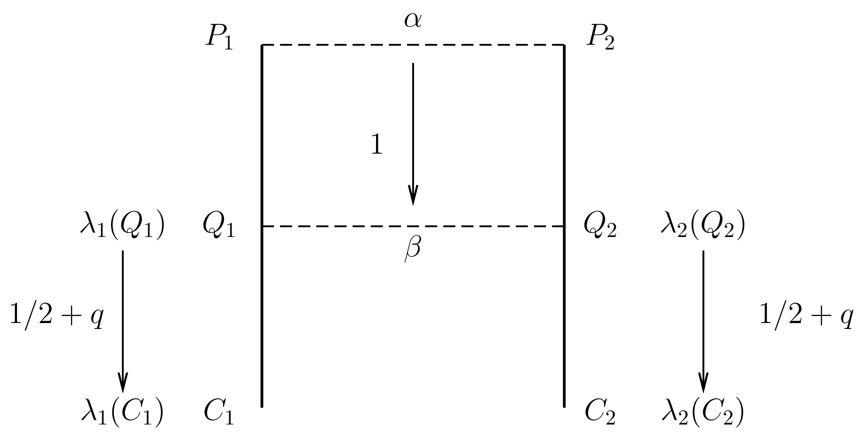


Figure 6.3: Differential-linear attack.

6.3.2 Other improvements

The differential-linear transition can be further generalized [BDK05a]. Assume that with probability 1 internal states form a balanced multiset of size m, i.e. all the states guaranteely sum into a fixed value (Section 4.1). Then the linear approximation of E_1 preserves this property, so we claim that $\bigoplus \lambda_C(C_i) = \bigoplus \lambda_1 Q_i = 0$ with probability $1/2 + 2^{m-1}q^m$. This can be proved by induction.

There were several attempts to decompose the primitive into three independent parts and more, but with no practical applications. It was conjectured [BDK05a], that any such decomposition can be transformed into a simpler attack with lower complexity.

6.4 Impossible differentials

If a differential is used as a distinguisher, not only possible difference propagations can be used but also impossible ones. Indeed, assume that the difference Δ_1 never propagates to Δ_2, which is called an *impossible differential*:

$$\Delta_1 \overset{f}{\not\to} \Delta_2.$$

If this holds for any key, the key recovery attack is mounted as follows.

Assume that an impossible differential is followed by a round with a key addition. Then we encrypt many plaintext pairs with the difference Δ_1, guess the last key (or a part of it) and partially decrypt ciphertexts. If the guess is correct, no pair of internal states has the difference Δ_2. Otherwise, the key guess is discarded. The attacker may guess on both ends of the primitive (recently used in [LDKK08]), though the filtering procedure is more complicated.

The differential $\Delta_1 \to \Delta_2$ must have high probability in a random permutation, otherwise the filtering procedure is weak. This is overcomed by making Δ_2 a truncated difference, which leads to a higher probability of the differential. However, a truncated impossible differential is harder to find.

Let us discuss the search for impossible differentials. An event with probability 0 implies that the complementary event has probability 1. Intuitively, it is difficult to find any property that always holds. One method is to find a trail on the number of rounds that do not provide full diffusion. Then we prove that particular output bits are not influenced by the input difference and, therefore, must have zero difference [WZF07]. Another method is to use truncated differentials, whose propagation can be investigated more efficiently. For example, one may distinguish only between zero and non-zero differences, or zero, non-zero, and arbitrary ones. Then linear and bytewise non-linear transformations are simple boolean functions, and all the impossible truncated differentials can be found by exhaustive search [WLSL10, LSL10]. Still, the theory lacks of necessary conditions to have an impossible differential.

If two sequential probability-1 trails are used:

$$\Delta \to \Delta'; \qquad \nabla' \leftarrow \nabla.$$

it is called the *miss-in-the-middle* [BBS99]:

$$\nabla' \neq \Delta' \implies \Delta \not\to \nabla.$$

The length of probability-1 trails is clearly limited by the diffusion properties of a primitive. As a result, the number of rounds covered by the impossible differential can be bounded by two times the number of rounds needed for full diffusion. The best attacks on AES-128 in a single-key model are the impossible differential attacks with truncated differentials. For all the versions of AES the first part of the impossible differential has the form $1 \to 4 \to 16$, and the second part can be of the form $12 \to 12$ [BA07, LDKK08] (Figure 6.4). The impossible AES differential can be preceded by a $4 \to 1$ differential with probability 2^{-24}. Recent impossible differential attacks on AES [MDRMH10] use sophisticated techniques, long outer differentials, and key schedule properties to mount the attack.

We also note that there are constructions, whose diffusion is much weaker in one direction than in the other one. For example, the diffusion of the AES key schedule is weaker in the backward direction ([FKL+00, BKN09]). Generalized Feistel schemes may have unbalanced diffusion as well.

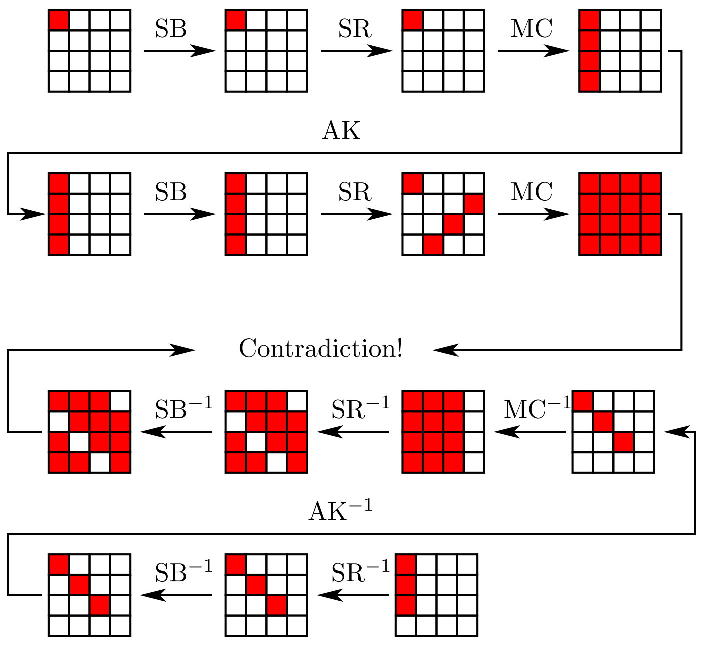


Figure 6.4: AES impossible differential.

Low-probability differentials and other extensions.

Possible differentials that have probability p_0 lower than expected from a random function (p), has been proposed since late 1990s [KR99], but their use is pretty limited. The reason is that the divergence $D(p||p_0)$ between two Bernoulli distributions is very small, so the data complexity of the low-probability differential attack is significantly higher compared to the impossible and regular differential attacks. As a result, the probability p must be relatively high, like in the impossible differential attack, should the low-probability attack be valid. A couple of low-probability attacks has been introduced recently [Tez10, MDS10], so we may expect further progress in that direction.

An analogue from linear cryptanalysis is a linear hull with zero correlation. Though hulls with these properties have been found for a reasonable number of rounds in AES and generic Feistel ciphers [BR11], they are far more difficult to exploit compared to impossible differentials. The data complexity is lower bounded with 2^{n-1}, i.e. the half of codebook.

6.5 Multi-branch functions

A typical example of double-branch function is the hash function RIPEMD, whose compression function operates as follows. The chaining value is input to two blockcipher-based permutations which use the message as the key. A new chaining value is the sum of two outputs:

$$CV_{i+1} = E_M(CV_i) \oplus E'_M(CV_i).$$

In order to simplify the design, the permutations differed in round constants only. Grøstl is an example of a wide-pipe double-branch, which works with fixed permutations P and Q:

$$CV_{i+1} = P(CV_i \oplus M) \oplus Q(M).$$

LANE has 6 branches.

The analysis of double-branch functions is usually diffuclt since it is hard to control both branches in parallel, which is required in most collision and preimage attacks. Cryptanalysts have designed several ad-hoc approaches to mitigate those difficulties.

If the permutations are fixed, like in LANE and Grøstl, an adversary may work in the semi-free-start framework. She finds a set of inputs with specific properties for each permutation, and then combine them to get an admissible input to the full compression function. Rebound attacks on LANE [MNPN+09] and Grøstl [MRST09] are typical examples. Since the condition for permutation inputs to be a right compression input is linear, the generalized birthday attack [Wag02] is a useful trick. This approach leads to collision and distinguishing attacks.

Another idea is to consider the difference between internal states of permutations ($P_{\text{round }r} \oplus Q_{\text{round }r}$ in Grøstl) while the message value is the same. Then the differential trails are caused by the difference in the operations in permutations, e.g. the constant addition, and not by the difference in the message. The first version of Grøstl admitted suitable differential trails similar to that in the first rebound attacks. The permutations P and Q differed only in the byte position of the constant addition. Then we assume that an internal state in round r is identical in both permutations, which is equivalent to the zero difference in a differential attack. Finally, we apply the rebound attack with a trail with non-zero difference injections at the positions of the constant addition [Pey10].

Regarding preimage attacks, the splice-and-cut method has been adapted for double-branch hash functions that are based on block ciphers. The idea is to pursue the meet-in-the-middle computation through the wrap-around in the same way it works for the Davies-Meyer mode ([SA09b, WSK+11], Section 5.2). We start with a state S in the permutation E, derive $CV = f(S)$ and use it in the permutation E'. Similar approach has been applied to the Grøstl hash function [MRST10].

Chapter 7

Representation and structure of a primitive

In this chapter we consider attacks that change or exploit the high-level properties of a primitive, or its *structure*. Some of these attacks change the *representation* of a primitive, i.e. provide an alternative view on its design, which simplifies the analysis and may demonstrate a weakness.

Contents

7.1 Slide attacks

Slide attacks were proposed in 1999 [BW99] against block ciphers with identical round transformations and subkeys. Assume that the encryption process admits the decomposition:

$$E_K = f_K \circ f_K \circ \cdots \circ f_K.$$

Consider any pair of plaintexts (P_1, P_2) such that $P_1 \overset{f_K}{\to} P_2$ (*slid pair*). Then $E_K(P_1) \overset{f_K}{\to} E_K(P_2)$. Due to the birthday paradox, we find a slid pair among about $2^{n/2}$ encryptions. A naive approach would be to test all the possible pairs (P_1, P_2) and then check an admissible key by the slide property in the corresponding ciphertext pair. However, the function f_K is supposed to be weak, so we can detect a slid pair faster.

The function f_K is not necessarily the round function, but can be any weak transformation that is repeated in a cipher. The complementation slide technique [BW00] deals with the function

$$f_K \equiv h_{K_2} \circ g_{K_1}.$$

Round constants, which slightly modify the round function for little cost, are a typical and easy countermeasure. As a result, few real ciphers were broken with this technique, but there appeared several notable applications.

Applications. Although SHACAL-1 uses four different constants (each for 20 consecutive rounds), it is vulnerable to the slide attack [Saa03]. The transitions between different round groups can be treated with a careful choice of the internal state. As a result, a slid pair can be constructed for the full SHACAL-1.

The KeeLoq cipher does not use round constants and was attacked with slides. The idea was to detect fixed points of the internal transformation (similarly to the reflection attack) [CBW08], or to guess a part of the key, decrypt the last rounds, and apply the guess-and-determine procedure [Bog08, IKD+08].

The simple prefix-MAC $E_K(M) = H(K||M)$ is vulnerable to the slide attack, if the underlying hash function consists of identical transformations. An example is the attack [GLP08] on the prefix-MAC based on Grindahl [KRT07], the stream-based hash function. Having processed the whole message, Grindahl performs several blank rounds, whose round function P is weak, and then truncates the output. The attacker constructs a message pair $(M_1, M_1||M_2)$ such that the hashing of the suffix M_2 repeats the computation of corresponding blank rounds for M_1. As a result, for some state S

$$E_K(M_1) = \text{Truncate}(S); \quad E_K(M_1||M_2) = \text{Truncate}(P(S)).$$

The weakness of P leads to the full recovery of S, inversion of the blank rounds and even the information on the key. This attack can be also viewed as a length-extension attack.

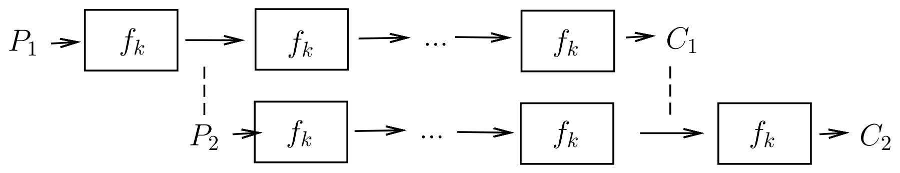


Figure 7.1: Slide attack.

7.2 Invariants

If a primitive E preserves a property $\mathcal{Q}$ of its input:

$$x \in \mathcal{Q} \to E(x) \in \mathcal{Q},$$

it is called an *invariant* for E. An invariant is a very dangerous feature of a design, even if it holds only with some probability. A slid pair can be viewed as invariant over a pair of inputs: the relation $x \leftrightarrow f_k(x))$ is preserved by f_k. The existence of two-input invariants is also related to the notion of homomorphic encryption [Riv02], which stands for deterministic transformation of algebraic relations over inputs. Homomorphic encryption is essential in the construction of security protocols such as voting, but is undesirable in the industrial-use block ciphers.

The use of round-dependent constants is a typical countermeasure against single-input invariants. Indeed, a round-dependent constant guarantees that round functions are different. A constant also has to break the invariant property $\mathcal{Q}$ to be a countermeasure.

7.2.1 Rotational cryptanalysis

The main idea of the *rotational cryptanalysis* is to consider pair of words where one is the rotation of the other one.

We define

$$\overrightarrow{X} \stackrel{\text{def}}{=} X \lll_r .$$

and call $(X, \overrightarrow{X})$ a *rotational pair* [with a rotation amount r]. A rotational pair is preserved by any bitwise transformation, particularly by the bitwise XOR and by any rotation:

$$\overrightarrow{X \oplus Y} = \overrightarrow{x} \oplus \overrightarrow{y}, \qquad \overrightarrow{x} \ggg_{r'} = \overrightarrow{x \ggg_{r'}}.$$

It also propagates through the addition modulo 2^n with probability between $1/4$ and $1/2$ [Dau05]:

$$\mathbf{P}(\overrightarrow{x+y} = \overrightarrow{x} + \overrightarrow{y}) = \frac{1}{4}(1 + 2^{r-n} + 2^{-r} + 2^{-n}).$$

The same holds for rotations to the left.

Now consider an arbitrary scheme $\mathcal{S}$ with additions, rotations, and XORs over n-bit words, q operations in total. Then with probability $(p_r)^q$

$$\mathcal{S}(\overrightarrow{I}) = \overrightarrow{\mathcal{S}(I)}.$$

under some independency assumptions.

For a random function $\mathcal{P}$ that maps to Z_2^t the probability that $\mathcal{P}(\overrightarrow{I}) = \overrightarrow{\mathcal{P}(I)}$ for random I is 2^{-t}. Therefore, we can detect nonrandomness if a function can be implemented with q additions, and $(p_r)^q > 2^{-t}$.

The main countermeasure against rotational cryptanalysis is the use of constants. If a constant is not rotation-symmetric, it always generates an error in a rotational pair. Low-weight errors can be cancelled due to non-ideal behavior of the modular addition.

Rotational cryptanalysis was used in the cryptanalysis of block ciphers Threefish [KN10, KNR10] and SEA [SPGQ06], and recently in attacks on BMW [NPSS10], Shabal [Van10], and ESSENCE [BDLF10]. It can be also applied for ciphers with bit-sliced S-box, like a modified Serpent [DIK08].

Like differential attack, rotational attack can be enhanced with neutral bits, auxiliary paths, and related tools (Section 8.5).

7.2.2 Fixed points

The notion of a fixed point is very important in the analysis of iterative transformations with identical round functions. If $f(P) = P$ then P is a fixed point for f. Fixed points are extensively used in the generic preimage attack [KS05], which are out of our scope. We will consider low-level attacks that exploit the existence of fixed points.

Reflection attack

In the reflection attack [Kar08], applied to the weak-key variant of GOST, a cipher is decomposed into a sequence of transformations: $F = f_l \circ f_{l-1} \circ \cdots \circ f_1$ such that all the f_i have the same fixed points. Then such a fixed point is also a fixed point for F.

The attack works as follows. Find a plaintext P that produces a ciphertext $C = P$, and then solve any equation $f_i(P) = P$, which is assumed to be relatively easy.

The fixed points obtained in the reflection attack can be also used in the meet-in-the-middle attack (Section 5.1). Assume that a random plaintext P is a fixed point with probability p for a transformation T_r made out of the first r rounds. Then an attacker runs the meet-in-the-middle attack on $1/p$ pairs $(T_r(P), C)$, where C is the ciphertext of P. As a result, the attack essentially skips the first r rounds no matter which neutral words are involved in these rounds [Iso11].

Collision search

Fixed points can be used in iterative compression functions in order to reduce the computational complexity of the collision search (the attack on the GOST hash function [MPR+08]).

An attack on RC4-hash is another example [IP08]. Assume that there exist two fixed points X and Y such that it is easy to find a path from X to Y. Then the following path produces a collision:

$$X \xrightarrow{M_3} Y \xrightarrow{M_2} Y$$
$$X \xrightarrow{M_1} X \xrightarrow{M_3} Y$$

7.2.3 Other invariants

A recent attack on Lesamnta uses invariants and bypasses the round constants [BDLF10]. It has been found that the function F in the generalized Feistel scheme preserves the swap of input halves:

$$\overleftrightarrow{A||B} \overset{\text{def}}{=} B||A; \quad F(A||B) = \overleftrightarrow{F(\overleftrightarrow{A||B})} = \overleftrightarrow{F(B||A)}.$$

The input to the function F is XORed to a round constant R_i, which was supposed to break this effect. However, the round constant are almost symmetric:

$$\forall i \; \overleftrightarrow{R_i} = R_i \oplus 1||1.$$

Then the attacker compares the compression of X and $\overleftrightarrow{X} \oplus 1||1$. The difference $1||1$ cancels the non-symmetry of constants, and each invocation of F gets a swapped input. Therefore, this attack demonstrates a combination of symmetrical invariant and a differential approach.

Now we briefly list the other applications of invariants in cryptanalysis.

- An differential of form $\Delta \to \Delta$ is an invariant over a pair of inputs and may form an iterative characteristic:
$$\Delta \to \Delta \to \cdots \to \Delta.$$
The first attack on the full DES [BS92] exploited an iterative characteristic.

- The complementation property of DES:
$$E_{\overline{K}}(\overline{P}) = \overline{E}_K P$$
is a probability-1 invariant. However, due to its deterministic nature, it only speeds up the exhaustive key search by a factor of 2.

- The complementation property of DES is actually the related-key differential with probability 1. A similar differential with non-one probability was constructed for GOST [KHL+04] and XTEA [BDLF10].

- The rotation of columns is invariant to the AES round function. While the round constant in the key schedule breaks this invariant in AES; it is not the case for several AES-based functions [BDLF10, DKS11]. Similar rotation property holds in Essence [BDLF10] due to the use of LFSR inside the compression function.

- The CubeHash round function preserves several symmetry patterns [ABM+09]. The use of such patterns is prevented by a non-symmetric IV.

- The SIMD compression function is vulnerable to various swap patterns despite the sophisticated message expansion [BFL10].

7.3 Fix and guess: simplifying the primitive

7.3.1 Block ciphers and hash functions

A primitive can be described as a system of nonlinear equations on its internal variables. Key recovery, preimage and collision search can be viewed as solving systems of such equations. These systems can be significantly simplified if we fix or guess some internal variables in advance, i.e. we search only for those executions whose internal variables satisfy particular conditions. Then an attack consists of two steps: produce an execution with desired properties, and exploit simplified equations. The technique is well known in the attacks on stream ciphers as the *guess-and-determine*, where it is primarily used to recover a internal state from a keystream. In block ciphers and hash functions this method is an auxiliary tool for more sophisticated attacks.

Key recovery. Most block ciphers have a strong round function, so that it is hard to recover the state and the key by guessing or fixing a smaller portion of internal variables. However, if the round function is weak, the technique is applicable. For example, a block cipher KeeLoq, which is based on a LFSR and thus resembles a stream cipher, was attacked with guessing the internal variables in a slid pair [Bog08].

Differential attacks. A fix of internal variables may limit the diffusion of difference and allow differential paths with higher probability. For example, consider a generalized Feistel scheme, which is similar to the MD family of hash functions:

$$A_{\text{new}} = A\&B \oplus C \oplus M_{\text{new}}; \quad B_{\text{new}} = A; \quad C_{\text{new}} = B,$$

where & is the bitwise AND. It is quite easy to construct a differential path that holds wtith probability 1:

ΔA	ΔB	ΔC	ΔM
0	0	$-\Delta$	Δ
0	0	0	0
0	0	0	0
0	0	0	0
Δ	0	0	Δ

A more flexible path can be obtained due to a local collision by fixing some internal variables to zero. Assume that $B = 0$. Then

$$A_{\text{new}} = C + M_{\text{new}},$$

so any difference in A does not propagate directly to A_{new}, but only to B. By fixing one more internal word to 0, we get a local collsion for M_1 and M_4:

A	B	C	M
0	$*$	$*$	$*$
$*$	0	$*$	$*$
0	$*$	0	$*$
$*$	0	$*$	$*$

ΔA	ΔB	ΔC	ΔM
0	0	0	Δ
Δ	0	0	0
0	Δ	0	0
0	0	Δ	$-\Delta$

Another example is the attacks on AES and AES-based hash functions [KBN09, BKN09]. We fix particular state bytes to ensure the difference propagation in the most expensive part of a differential trail. Then we find a conforming execution efficiently and check whether it conforms the rest of the trail (Figure 7.2). Similar ideas are used in the rebound attack (Section 4.3) the attack on SHAMATA [IMPS09].

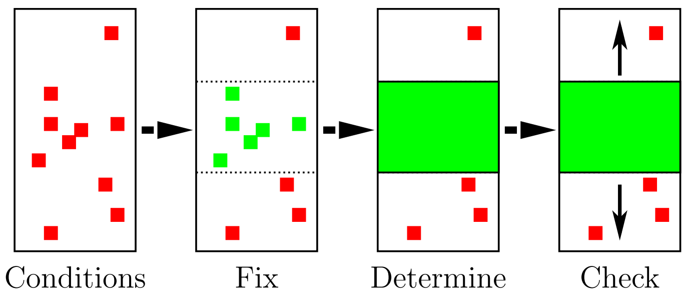


Figure 7.2: Fix, determine and check.

In collision search. Bit-oriented hash functions like the SHA family are vulnerable to collision attacks where a part of the IV is fixed to values that enhance the collision search. If the adversary is limited to the complexity 2^t, she may fix up to t bits in the intermediate IV just by random trials.

This technique was used in the attack on the compression function BMW [Tho10]. The attacker fixes several internal variables to zero, so that the resulting hash value is almost equal to the message block. Therefore, a flip of a single bit results in a pseudo-collision for the compression function.

A more advanced technique is applied to hash functions Lesamnta and SHAvite-3, which are based on the generalized Feistel scheme. By equalizing outputs of the Feistel function F, the attackers simplified the equations that describe the hash value [BDLF09, BDLF10].

Using auxiliary paths (Section 8.5) in differential attacks, the attacker explicitly fixes some internal bits in order to increase the probability of the path. For example, by fixing one bit to zero in the argument of the modular addition, one can guarantee that there is no carry in this bit.

In preimage search. This technique under the name “absorption” was used in several preimage attacks on hash functions of the MD-family. For example, if two of three inputs to the majority function are equal to C, then the output is always C regardless of the third input. Therefore, any difference in the third input word would be absorbed, and a local collision property, that involves that word, would hold with higher probability. This principle was extended to other nonlinear functions and resulted in preimage attacks on HAVAL, MD4, and MD5 [Leu08, SA08a, SA09a].

The guess-and-determine technique can be applied to non-invertible transformation to make it invertible and, thus, vulnerable to such attacks as the meet-in-the-middle (Section 5.1). For example, the Edon-R compression function can be inverted by guessing particular words in the internal state [KNW09]. Early designs can be attacked directly by the guess-and-determine, such as MD2 [Mul04].

Extremely short compression functions are vulnerable to guess-and-determine preimage attacks with Hamsi as a notable example [Fuh10]. Dinur and Shamir significantly reduced the pseudo-preimage search by fixing several bits of the IV.

7.3.2 Stream ciphers

The fix, or guess-and-determine method is extremely popular in the analysis of stream ciphers, possibly because of the abundance of equations and variables that appear while observing a large amount of keystream. The guess-and-determine attack is used both for the internal state recovery and the key recovery.

Byte-oriented design are more vulnerable to the attack due to smaller number of variables and shorter equations. On the other hand, the non-linear transformations of the byte-oriented design are more complicated. Still, there is little theory behind the guess-and-determine attack, besides a trivial bound that it is possible whenever the CICO problem (Section 2.2) can be solved. Some stream ciphers have data-dependent transformations (e.g., irregular clocking), which are also subject to the guess-and-determine attacks [NTW10]. The papers [DG08, FLZ+10, HR02, MK08] may be the further references.

7.4 Ladder tricks: exploiting parallelism

The internal state of a primitive is a conjunction of low-level blocks, like bits or bytes. The diffusion principle states that the blocks must be mixed with each other in order to make every output bit dependent on all the input bits. Since the round function is not ideal, the internal states can be decomposed into independently processed blocks. This effect can be exploited in many ways.

7.4.1 In boomerang attacks

In the boomerang attack we decompose a cipher into two subciphers E_0 and E_1. The border between subciphers is not necessarily the border between rounds. Moreover, it does not have to be a border between operation layers, like S-box layers. In general, the border is a set S of internal variables, that completely determine the execution of the cipher in both directions. For example, the internal state A^3 (after S-box layer in round 3 of AES-128) and the subkey K^4 (the subkey of round 4 of AES-128) together determine the computations in both directions.

It was demonstrated that such simple slices may not be optimal in terms of the probability of the corresponding boomerang distinguisher. In boomerang related-key attacks on AES [BK09] this principle was called the *ladder trick*. For example, assume that the differential trail for E_0 has low number of active S-boxes in rounds 1–4, and the differential trail for E_1 has low number of active S-boxes in rounds 6–9. However, round 5 would have too many active S-boxes, if it were entirely assigned to any of the subciphers.

The solution is to extend both subciphers to the S-box layer in round 5, and then assign the S-boxes independently. We assign each S-box to the subcipher where it would be inactive, thus greatly reducing the total number of active S-boxes in the boomerang trail.

7.4.2 Partial matching

The same principle in preimage search got the name “partial matching”. In the meet-in-the-middle preimage attacks (Section 5.2) two computation chunks are to meet at some intermediate step of the compression function. However, the chunks may not be computed independently till the meeting point, because the next step involves a neutral word used in the other chunk. Then we notice that the size of the neutral word is often smaller than the size of the state, so the state can be partially computed even after the point of injection. As a result, we match the states partially [AS08, SA09a, AGM+09]. Work on a bit level admits better results, since the diffusion is slower, and the effect of the message injection depends on the bit positions.

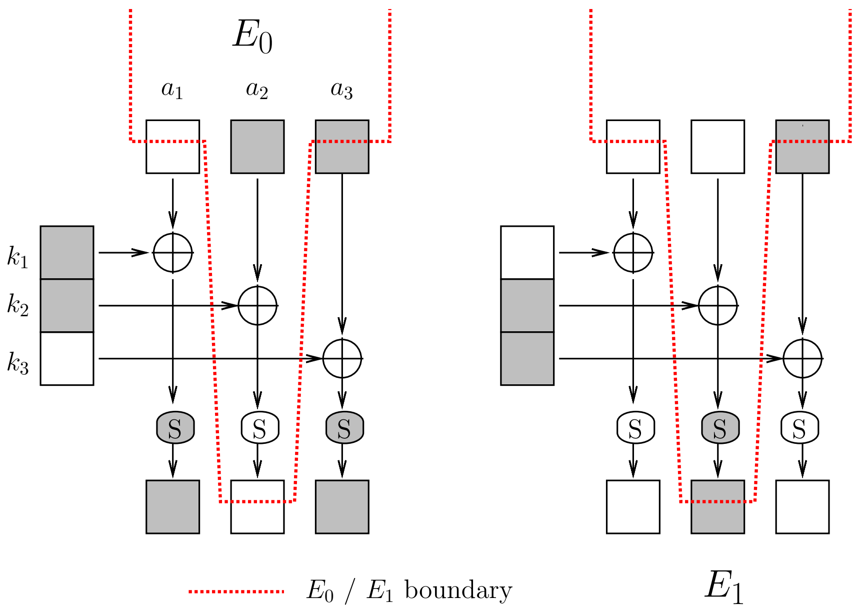


Figure 7.3: The ladder switch in a toy three S-box block. A switch either before or after the S-box layer would cost probability, while the ladder does not.

Chapter 8

Proper use of probabilistic patterns

This chapter is devoted to various issues related to the probabilistic patterns, most of which were introduced in Chapter 3. Recall that a probabilistic pattern for a function f is defined with respect to the input property $\mathcal{A}$ and the output property $\mathcal{B}$, which is derived with probability p:

$$\mathbb{P}\left[\mathcal{A} \xrightarrow{f} \mathcal{B}\right] = p.$$

A pattern with probability higher than what is expected in a random function, it is a potential weakness in an n-bit permutation (or transformation). There are several common approaches to convert such a pattern to an attack, depending on the attack nature.

Contents

8.1 Find a good pattern

The search for a good pattern is typically a combination of manual and automated work, and the manual efforts prevail for good designs. The first long characteristics were *iterative*, i.e. they were a repetition of a single pattern of form $\alpha \rightarrow \alpha$ [BS92]. However, this approach fails for modern primitives due to better diffusion properties.

Linearization. The ARX primitives are subject to linearization, i.e. to the replacement of all modular additions by XORs and sequential search for low-weight trails. If found, those trails serve as a base for differential attacks. The nonlinear paths are missed, so this approach does not guarantee the absence of high-probability trails. On the other hand, nonlinear trails are quite difficult to handle. As a result, cryptanalysts prefer nonlinear trails that sligthly deviate from the linear ones [MN09, KNR10].

Greedy algorithms. SHA-1 and CubeHash trails close to optimal have been found by the combination of the linearization technique and a greedy algorithm [CR06,KKMS10]. The idea is the following. First, the majority of the internal state bits of a trail are marked as "undefined". Then they are sequentially defined in a way that produces the differential trail most suitable for the collision attack (in the middle and the last rounds this is equivalent to getting the highest probability). The search is accompanied with numerous heuristics and does not guarantee the absence of better trail.

SAT-solvers and similar. SAT solvers and tools as Gröbner basis are often viewed as universal solutions to cryptanalytic problems. However, due to the nature of cryptanalysis such generic tools make sense mainly when doing some mechanical work like searching or testing probabilistic patterns. The advantage of using these tools is still quite limited, and the crypto community remains sceptical about most of the attacks mounted in this framework ([MZ06] is a nice though theoretical exception). Several attempts to prove the relevance of these tools by designing a cipher vulnerable only to them failed [DK09].

Branch and bound. The branch-and-bound strategy is applied when the problem can be reformulated as a search over a tree of possible solutions. For example, differential trails for collision search in a hash function form a tree that starts with zero-difference state and split into branches in non-linear operations. Every operation is a node which has as many branches as many differences may come out of the operation. If a branch is marked with a logarithm of the corresponding differential probability, then the path from the root to a leaf with the largest sum of marks gives the highest probability trail.

The branch-and-bound strategy optimizes the search by cutting out those paths that will not lead to an optimal solution. For this the strategy needs a lower bound on the value of the optimal solution. Let us denote by V_r^* a lower bound for the optimal solution over r rounds, and by $\overline{V}_r$ an upper bound for the optimal solutions over r rounds. Assign V_1 with the value of the best one-round solution that we can find. Derive $\overline{V}_1$ from the structure of the primitive or choose the maximum possible value.

Now suppose we have a lower bound V_n^* for the optimal solution over $n > 1$ rounds (i.e., by taking an arbitrary solution), and the upper bounds $\overline{V}_i$ for the optimal solutions over $i < n$ rounds. Then we construct optimal n-round solutions as follows. First, consider only those solutions whose value v_1 in the first round satisfies the condition

$$v_1 \cdot \overline{V}_{n-1} > V_n^*.$$

Evidently, the other solutions have values lower than V_n^* if being extended to n rounds. Then extend the remaining solutions to two rounds and consider only those extensions whose values v_2 in the first two rounds satisfy the condition

$$v_2 \cdot \overline{V}_{n-2} > V_n^*.$$

Then proceed with the third round and so on until we construct the best solutions for n rounds.

The branch-and-bound strategy is exponential in the worst case. Indeed, assume that there are q optimal one-round solutions which are compatible with each other. Then after r rounds we have q^r optimal r-round solutions. However, if there are few optimal one-round solutions and they are mainly incompatible, then the number of solutions might remain relatively small. The recent attacks on AES, DES, and other ciphers [BN10, BN11] exploit the fact that there are few optimal one-round trails in these primitives, and the previous estimates of V_n^* are close to optimal.

8.2 Key recovery

The key recovery is the primary attack on a block cipher, and a probabilistic pattern is well suited for this purpose. The attack procedure depends on if the pattern probability is determined by a single (dominant) trail or a cluster of trails resulting in a hull or differential.

8.2.1 Dominating trail

If the attack pattern is dominated by a single trail, the key bits can be derived explicitly from the right pairs. Assume we have a pattern $\mathcal{A} \xrightarrow{f} \mathcal{B}$ with probability $p \ll 1$, which in turn is much higher than the proportion of elements with property $\mathcal{B}$ in the range of f. Then we collect a right input after about $1/p$ random trials. Since we know the property of the states before and after the last key addition in f, we are able to get information on K.

For example, assume that a nonlinear transformation f is applied to the internal state in the last round, and then the key is XORed to get the ciphertext:

$$C = f(A) \oplus K.$$

Suppose we work with a differential pattern and know the differential of f: $\Delta_I \xrightarrow{f} \Delta_O$, then we get the following equation:

$$f(A \oplus \Delta_I) = f(A) \oplus \Delta_O,$$

which provides information about $f(A)$. Due to the nonlinearity of f and the awareness of C, we get a list of candidate keys $\{K\}$. We use a simple rule of thumb: key candidates are ranked according to the number of times they are proposed by ciphertexts.

The simplest linear attack (Algorithm 1) deals with a single linear trail, which approximates a linear function of subkeys $l(\overline{K})$ with $(u \cdot x \oplus w \cdot f(x))$ for some u, w. The value $l(K)$ is set according to the dominate value in the samples $\{u \cdot x_i \oplus w \cdot f(x_i)\}$. The data complexity is proportional to $1/C^2$, where C is the correlation of the dominating trail. It was later demonstrated [Jun05] that the simple rule is optimal in terms of the Neyman-Pearson lemma. This result does not hold [Jun05], however, for the case of multiple approximations, where the formal approach from the nest section makes sense [HN11].

8.2.2 Multiple trails

In this section we explain an approach with a distinguisher, which is necessary if there is no dominating trail. It is also useful in the other case, since it allows to recover key bits sequentially. Now we assume that a probabilistic pattern $\mathcal{A} \xrightarrow{f} \mathcal{B}$ covers all but r_0 last rounds. Typically, r_0 is 1 or 2; it may also cover parts of rounds. Then we guess the part of the last subkeys that is necessary to figure out from the ciphertexts C whether the property $\mathcal{B}$ holds for the relevant proportion of intermediate states $S = f(P)$. The right guess should show that the proportion is about p_0, where p_0 is the probability of the pattern. A wrong guess should provide a value significantly different from p_0.

Therefore, we face a purely statistical problem. It would admit a simple solution if we were not restricted in time. Since we seek the fastest algorithms, the procedure that determines the key may be quite complicated. Statistical tools used in the attack determine how efficiently we use the data, whether we are able to exploit multiple patterns simultaneously, etc. The task is most difficult in linear cryptanalysis (see [BCQ04, HN11]).

Now we proceed with a formal explanation, which is substantially based on the research by Blondeau, Gerard, and Tillich [BG09, BGT11]. Let N be the number of available plaintext/ciphertext *samples*. A sample can be a single pair (linear cryptanalysis), two pairs (differential cryptanalysis), etc.. Generally, we get information not on the full key, but on a subkey. For that we extract some *statistic* Σ from the available data, then compute the likelihood of each possible subkey, suggest a list $\mathcal{L}$ of the likeliest keys, and try exhaustively all the corresponding master keys until the correct key is found.

In most of the cases, the statistic Σ is a set of counters Σ_k that correspond to the number of times the probabilistic pattern is observed for a subkey k. Let p_0 be the probability that the pattern is observed for the correct subkey k_0, and p be the probability that it is observed for an incorrect subkey. Here we assume that all the incorrect subkeys have the same pattern probability, which is a standard assumption. Without loss of generality, let $p < p_0$. The likelihoods are just the counter values.

Assuming also that the samples are independent, the counter Σ_k follows a binomial law with parameters (N, p) for $k \neq k_0$ and (N, p_0) for $k = k_0$. Then the attacks can be mounted in two different frameworks:

- Fix a threshold and accept the keys with a likelihood larger than the threshold (*hypothesis testing*).
- Fix the size of $\mathcal{L}$ to some value l and keep the l likeliest subkeys (*key ranking*).

Hypothesis testing

We fix threshold T and accept subkey k if $\Sigma_k \geq T$. Then the success probability P_S of the attack is determined by the probability $\alpha = 1 - P_S$ of the type I error, which occurs when k_0 is not accepted due to $\Sigma_{k_0} < T$. The time complexity of the attack is determined by the size of $\mathcal{L}$ and thus by the probability β of the type II error, which occurs when $k \neq k_0$ is accepted. The well-known result from statistics states that no test can improve the threshold test in both α and β error probabilities.

Denote T/N by τ. If $p < \tau < p_0$, the following asymptotic holds for the error probabilities [BGT11]:

$$\beta = \mathbb{P}\left(\Sigma_{k \neq k_0} \geq \tau N\right) \xrightarrow{N\to\infty} \frac{(1-p)\sqrt{\tau}}{(\tau - p)(\sqrt{2\pi N(1-\tau)})} e^{-N \cdot D(\tau || p)}; \tag{8.1}$$

$$\alpha = \mathbb{P}\left(\Sigma_{k_0} < \tau N\right) \xrightarrow{N\to\infty} \frac{p_0\sqrt{1-\tau}}{(p_0 - \tau)(\sqrt{2\pi N\tau})} e^{-N \cdot D(\tau || p_0)}. \tag{8.2}$$

$$\tag{8.3}$$

Here $D(p||q)$ is the *Kullback-Leibler divergence* for Bernoulli distributions:

$$D(p||q) = p \ln\left(\frac{p}{q}\right) + (1-p)\ln\left(\frac{1-p}{1-q}\right).$$

Therefore, time and data complexities and the success probability are all related, and any two of them determine the other one. Since the equations (8.1) are hard to invert, the data complexity as a function of α and β is found by numerical methods.

For the threshold $\tau = p_0$ and the resulting $\alpha \approx \frac{1}{2}$, data complexity may be approximated as follows:

$$N' = \frac{\ln(2\sqrt{\pi}\beta)}{D(p_0||p)}. \tag{8.4}$$

The threshold may be also given by the log-likelihood ratio or the convolution method [HN11]. If the value of p is not available, one may use the χ^2 test.

Key ranking

Within the key ranking approach we have to fix the number l of likeliest keys in order to try them in the next phase. The value of l is a tradeoff between the success probability of the attack and the time complexity. We sort the subkeys according to the counter value Σ_k and store those with the largest counter value. The success probability is then defined as

$$P_S = \mathbb{P}[k_o \in \mathcal{L}].$$

Let n be the total number of subkeys. The explicit formula for P_S in terms of l and n is quite sophisticated, so we provide an approximation, which was checked experimentally for different types of patterns [BGT11]:

$$P_S \approx \sum_{i=F^{-1}(1-\frac{l-1}{n-2})}^{N} f_0(i),$$

where F^1 is the inverse of the cumulative function of a binomial law with parameters (N, p):

$$F(x) = \sum_{i \leq x} \binom{N}{i} p^i (1-p)^{N-i},$$

and $f_0(x)$ is the probability that the counter Σ_{k_0} is equal to x:

$$f_0(x) = \binom{N}{i} (p_0)^i (1-p_0)^{N-i}.$$

Data complexity for various cryptanalyses

It is a well-known rule of thumb, that exploiting a bias ε in a linear attack requires about $1/\varepsilon^2$ data, while a differential with probability ε is exploitable with only $1/\varepsilon$ plaintext pairs. One may ask why the situation is so different. To answer the question we look at the expression (8.4) and notice the crucial role of the Kullback-Leibler divergence. Computing the divergence for distributions appearing in linear and differential cryptanalysis is a simple exercise. The result [BGT11] is as follows:

Attack	Number of samples
Linear	$\frac{1}{2(p_0-p)^2}$
Differential	$\frac{1}{p_0 \ln(p_0/p) - p_0}$
Truncated differential	$\frac{p}{(p_0-p)^2}$
Impossible differential	$\frac{1}{p}$

It is easy to see that the data complexity of the differential cryptanalysis depends rather not on the difference between p and p_0 but on their ratio and the magnitude of p_0.

8.3 Attacks on compression functions

A probabilistic pattern is a universal tool for the analysis of a compression function if it clearly exhibits nonrandomness of the primitive. A collision search is actually a search for inputs producing a zero-ending differential trail. Therefore, a good trail may form a base for a collision trail if it fits the attack framework (zero final difference in a collision, zero IV difference in a semi-free-start collision, etc.).

A zero-ending trail can be also used for a second-preimage search. Indeed, applying the specified difference to the original input, we get the same output with the probability equal to the DP of the trail. However, this method has very low probability to succeed (see the attack on MD4 [YWZW05] and Haval [LCK+08]), since every trial requires a new differential.

A non-zero ending trail can be a base for a multi-block collision (Section 8.4), especially when a feedforward is used. If $g(x) = f(x) \oplus x$ then a differential $\Delta \to \Delta$ for f is a differential $\Delta \to 0$ for g.

If the round function is not bijective, it is a dangerous weakness. For example, short zero-ending differentials (vanishing differentials) have been found in the SecurID hash function [BLP03].

MAC. Differentials are widely used in key recovery attacks on the message authentication codes. In this setting a MAC is equivalent to a block cipher, so the attack methods are very similar. Since many MACs are based on hash functions, the differentials are taken from collision attacks on these functions [FLN07, WOK08].

8.4 Search for conforming executions

In this section we discuss optimal search for executions that conform to a probabilistic pattern. We consider a function f as a black box, for which we are given a pattern with probability p:

$$\mathbb{P}\left[\mathcal{A} \xrightarrow{f} \mathcal{B}\right] = p. \tag{8.5}$$

The further text is relevant for patterns only with relatively low p, such as differential of rotational attacks. It is not suited for the linear cryptanalysis.

To find a pair of inputs conforming (8.5) one has to make about $1/p$ queries with sets with property $\mathcal{A}$. This is done with negligible memory and time complexity of about $1/p$ queries, where s is the size of a set with property $\mathcal{A}$. For the differential cryptanalysis we make $2/p$ queries with single texts.

Now consider a more general multi-step pattern, when we have some freedom to modify inputs between invocations of f_i:

$$\mathcal{A}_0 \xrightarrow{\text{Injection}} \xrightarrow{f_1} \mathcal{A}_1 \xrightarrow{\text{Injection}} \xrightarrow{f_2} \cdots \mathcal{A}_{k-1} \xrightarrow{\text{Injection}} \xrightarrow{f_k} \mathcal{A}_k.$$

This situation is typical in stream-based hashes. Denote the number of possible modifications at step i by 2^{l_i} and the number of bit conditions that must be fulfilled at each step by q_i.

Let us compute the complexity of the attack, i.e. of finding a conforming set of executions. Denote the number of starting points by N, then $N2^{l_1-q_1}$ points pass the conditions of the first step. Then this number is increased by 2^{l_2} modifications, thus giving $N2^{l_2+l_1-q_1-q_2}$ points:

$$N \xrightarrow{\text{Injection}} N2^{l_1} \xrightarrow{f_1} N2^{l_1-q_1} \xrightarrow{\text{Injection}} N2^{l_1+l_2-q_1} \xrightarrow{f_2} N2^{l_1+l_2-q_1-q_2} \ldots \xrightarrow{f_k} N2^{\sum_{1\leq j\leq i}(l_i-q_i)}.$$

Most attacks needs just one right point in the end. Together with the condition that at least one right point must come of each step, we have the following condition

$$\forall i \quad \log N + \sum_{1\leq j\leq i} (l_i - q_i) \geq 0. \tag{8.6}$$

In the attack we choose minumum N satisfying Eq. 8.6. Clearly, the attacker is not obliged to use all the degrees freedom at each step. Let us note that the sequence of intermediate points is *a path on a tree* from its root to the deepest leaf, where the root is a starting point, and branches are modifications, of which a proportion of $1 - 2^{-q_i}$ ends with a leaf.

The workload C of the attack is approximated by the number of right pairs entering the round, or the *width* of the search tree. Assuming that at least one right pair comes out of a round, we get the following approximation:

$$\log C \approx \max_{i_1\leq i_2\leq k}\left[l_{i_1} + \sum_{i_1\leq j\leq i_2} (q_{i+1} - l_i)\right].$$

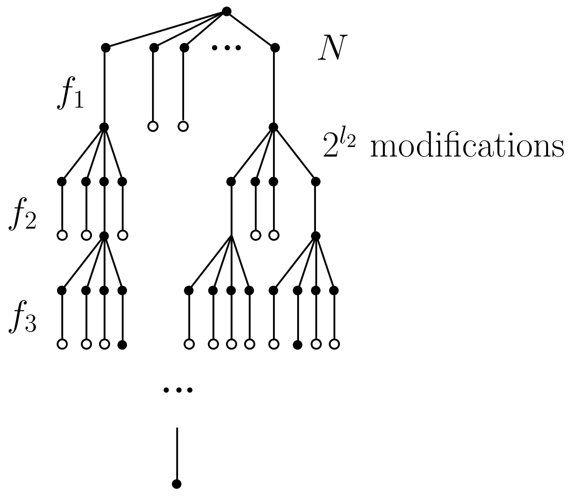


Figure 8.1: Tree of the search for conforming sets.

We described the search procedure as we first computed all the points at a particular step (layer of the tree) and only after checked whether they would lead to the next step (Figure 8.1). This approach, which is actually a *breadth-first search*, requires too much memory. The *depth-first search*, when we explore the tree as far as possible along each branch, is more efficient since we have to store in memory only the path of length k. In the differential cryptanalysis the depth-first search is known as *trail backtracking* [BDPA06].

Collision search inside a compression function. This framework is quite popular in the collision search in regular hash functions, such as SHA-1 [CR06, CMR07]. For example, to find the conforming message pair for the first block of the 64-step SHA-1 collision [CR06], one has to start with the standard IV (one possible pair), and then use 95 bit degrees of freedom to bypass 55 bit conditions in the first 16 rounds. If a condition is failed, the attacker backtracks the process and tries another possible input. Therefore, after 16 rounds about 2^{40} execution pairs are left, and they are filtered with the 40 uncontrolled conditions in the next 48 steps, leaving one valid collision pair. In our notation we have the following:

$$\begin{cases} \sum_{i\leq 16} q_i = 55;\ \sum_{i>16} q_i = 40; & N = 2^{40}; \\ \sum_{i\leq 16} l_i = 95;\ \sum_{i>16} l_i = 0. \end{cases}$$

If we work with truncated differentials, an l-bit injection may provide up to 2^{2l} modifications, depending on the power of the constraint. In the attack on Grindahl [Pey07], which injects 4 bytes at each step, Peyrin exploited differentials truncated to bytes. Therefore, he had 2^{8k} degrees of freedom at each step.

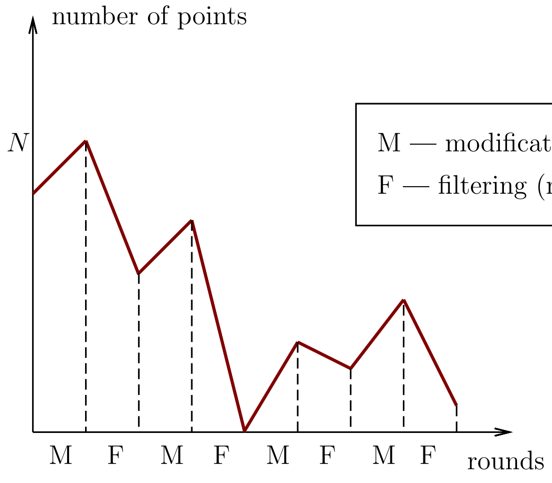


Figure 8.2: Complexity of trail backtracking.

Multi-block collision. A multi-block collision spans over several calls of a compression function, and can be found with the backtracking method. The differential characteristic between the calls of the compression function has the following form:

$$0 \xrightarrow{F} \delta_1 \xrightarrow{F} \delta_2 \xrightarrow{F} \cdots \delta_k \xrightarrow{F} 0.$$

If the compression function has a feedforward, the full collision is a concatenation of pseudo- and near-collisions, for which differentials with higher probability may exist.

For example, the first collision for SHA-0 was a four-block collision [BCJ+05], and the blocks were computed with complexities $2^{49.5}, 2^{50.5}, 2^{50.5}, 2^{44}$, respectively, with the total complexity around 2^{52}.

Use of structures.

A *structure* is a set of states where particular bytes take the same value. If input pairs can be organized into structures, the attack complexity can be reduced. Typically, a structure of size T has $T^2/2$ pairs (Figure 8.3). The most common difference type that allows such a composition, is the arbitrary difference. For example, a fixed difference in 12 out of 16 bytes corresponds to a structure of 2^{32} tuples (states), which can be composed into 2^{63} pairs. This may be enough to pass through a trail with probability 2^{-60}. Ideally, a trail with total differential probability 2^{-k} would be exploited only with $2^{k/2}$ data complexity.

Typical trails use regular differences throughout the primitive, and truncated differences only in the beginning or in the end. As a result, mainly structures of plaintexts and ciphertexts are used, so the advantage in complexity is limited.

However, the trails for hash functions are more flexible, so truncated differentials can be widely applied. The attacks on stream-based Grindahl and RadioGatun are the examples. These differential trails can be adapted for the use of structures, as was shown in [Kho09a]. On the other hand, the use of structures imposes a big constraint on the injected messages. Consider, e.g., a 3-byte message injection, in which the difference is fixed to zero in the first byte, and is unspecified in the remaining two bytes. Then all the elements of a structure that follow the trail should get the message with the same first byte, i.e. there are 2^{16} possible inputs per element. If we considered pairs separately, each pair would admit $2^{8+2\cdot 16} = 2^{40}$ pairs of injections. As a result, the structure splits faster than the number of pairs decreases, which means that the structure at some step simply fissions into separate pairs (Figure **??**).

*	C	C	C
C	*	C	C
C	C	*	C
C	C	C	*

Figure 8.3: Structure of 2^{32} AES states, which form 2^{63} pairs of texts. C stands for a constant, * stands for an arbitrary value.

8.5 Speeding up the attack

In this section we discuss several tricks for speeding up an attack with probabilistic patterns, mainly a differential attack. Recall that a conforming pair for a differential with probability p can be found after $1/p$ random trials. The main idea of the following method is to reduce the complexity of the search using the information from previous queries.

Block ciphers

Since we do not observe the encryption process, we are not able to detect why a pair with a right difference does not conform to a differential. As a result, we do not get information from such a failure.

If some sufficient conditions for the differential trail are simple functions of the key and the plaintext, the latter can be simply modified until the conditions are met [KMNP10, DS11a]. The more efficient way, however, is to encrypt a structure of plaintexts and retrieve the key bits from the right pairs. When a right pair is obtained, it may speed up the search for the next pairs if it is required. In the attack on RC5 [BK98b] the second and next right pairs can be obtained with significantly lower complexity. In the attack on AES a single right pair provides information on several key bytes, so the attacker partially controls the first round and get right pairs for the next differential much faster [BKN09].

Compression functions

In compression functions we observe the transformation process, so we know where a candidate pair leaves the differential path. There is a couple of tricks aimed to modify the execution so that it passes further. As a result, if an attempt fails, we hope to succeed in the next attempt with higher probability than in a random trial.

Message modification. The message modification is a procedure that tells how to change the inputs if a condition in a particular round is failed. The first such procedure [WY05] was proposed for MD5 was considered. The idea is not to discard a message pair but modify it in a special way. Clearly, the modification is easy if an error occurs shortly after an controlled injection (*basic modification*), but is hard if it occurs several rounds after the last controlled input (*advanced modification*). Several modification techniques were introduced, of which many are ad-hoc [SWOK07, NSKO05].

Auxiliary paths. In the attacks on MD/SHA family of hash functions a condition, that a message conforms to a differential, was translated to simple conditions on internal bit variables. For example, if a particular bit is 0, the difference passes the modular addition. However, it is quite hard to control bits if they are located far from the last controlled input. We note that in start-from-the-middle attacks a cryptanalyst can fulfill conditions in both directions, thus getting a more efficient attack [KNR10].

There appeared numerous techniques that aimed to bypass these conditions without discarding the current trial. Virtually all of them are dedicated and suited for only one primitive. In the further text we will briefly cover the underlying ideas.

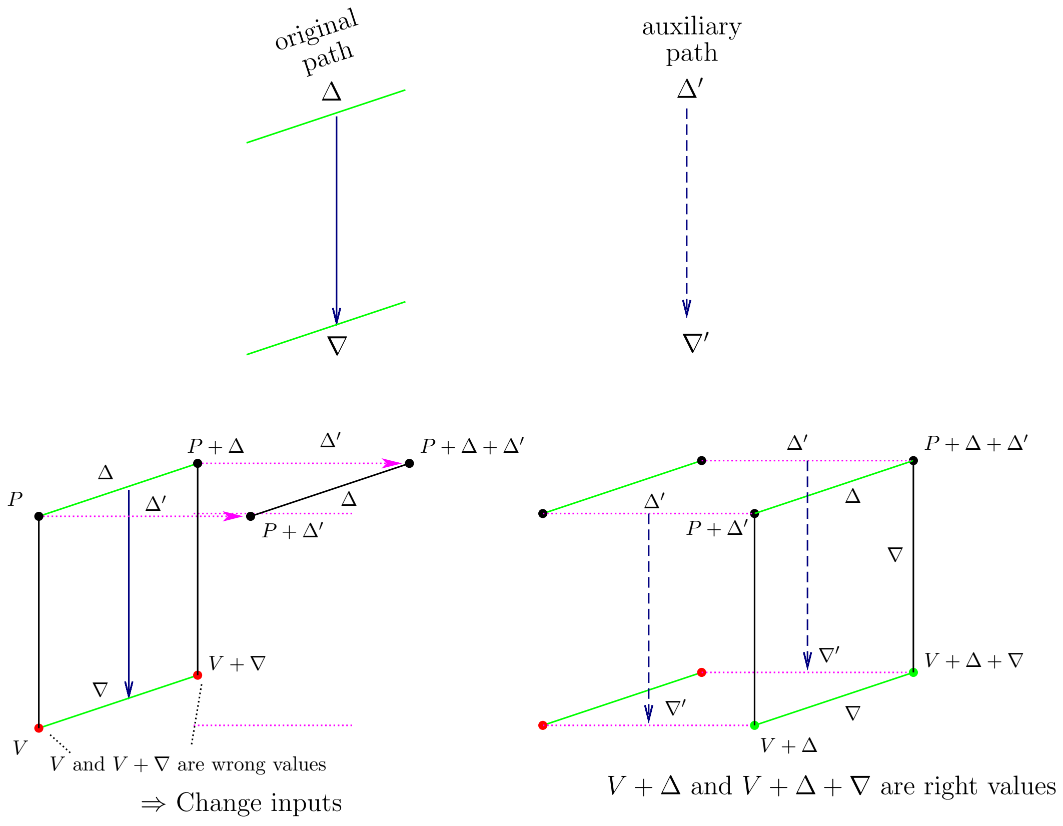


Figure 8.4: Auxiliary paths.

A bit b is called *neutral* with respect to an internal variable X if its flip does not affect

X. Certainly, there might exist also neutral bytes or words [BC04]. Then one can change the value of the neutral bit in order to affect only a particular variable in the further computation, and not to affect the others. Two neutral bits can quadruple the number of execution pairs conforming the trail. This method was later generalized to probabilistic neutral bits [AFK+08] and a generic framework for collision attacks on ARX primitives [BKMP09].

Authors of [BKMP09] proposed to split message into groups of bits according to their influence on particular internal variables. Then in the first step an adversary "plays" with message bits that influence some of the conditions, and in the next step these message bits are fixed, and the others are modified.

In an *amplified boomerang modification* [JP07] (also known as a difference δ is applied to both messages in a pair so that particular bits are changed in the middle of computation (Figure 8.4). Let us show the effect on the following example. Assume that a message pair must assign particular bits B in the computation to 0 in order to conform to the trail. However, in the current computation they are equal to 1. Let the message difference δ result in a differential trail, which produces difference in B with probability p_δ. Then we apply δ to both message words in our computation, so that B in both executions is changed to 1 with total probability p_δ^2. If the latter value is higher than the probability that a random message pair reaches this round, this method is useful. Notice that this attack relates to a boomerang attack on ciphers since it creates quartets of states, but the boomerang does not "return".

Simpler version of these ideas appeared under names *tunnel* [Kli06, Ste07] and *submarine* [NSS+06]. A generalization of all the techniques has also been described in Rechberger's PhD thesis [Rec09].

NL-part and Inside-out

It is easy to control the internal state in the first rounds, because internal variables can be easily changed by input variables. As a result, an attack algorithm may admit a more sophisticated treatment of the first rounds. In the SHA-x hash functions the first rounds are directly affected by the message words, so it is easy to fulfill more sophisticated conditions in the corresponding internal states.

As a result, the first part of a differential trail is a subject to a serious modification. In the case of SHA-x the cryptanalysts separate the first part of the trail (called *nonlinear*, or NL-part) from the second *linear* part. The nonlinear part is chosen so that the probability of the linear part is maximal. The number of conditions in the non-linear part is less important, since most of conditions can be fulfilled by a careful choice of input message words.

If the attacker is not restricted in the inputs of the transformation (like in a pure collision attack), he may first find solutions to the most expensive part of the differential trail (or another pattern), and use the remaining freedom to conform to the conditions in the outer phase. This approach is common in the rebound attack (Section 4.3) as well as in the attacks that do not use meet-in-the-middle principle to construct differential trails [NPRA+10].

8.6 Solving equations

Many attacks are reduced to the problem of solving non-linear equations. Though the generic problem is NP-complete [GJ79], a cryptanalyst hopes that a structure of a primitive makes it vulnerable to some heuristic or tool.

Ad-hoc. Solving equations manually is quite popular when the primitive (or the part to be attacked) is relatively small and equations are simple. Then a cryptanalyst may notice the properties that are missed by an automatic tool, or provide a slower but deterministic way to solve the system of equations. The advantage of this method is that a reader may check the solution himself without running the software, or figure out a method that is useful in another attack.

SHA-3 competition inspired dozens of new designs, and many of them are apparently so weak that attacks are mounted just by ad-hoc solving of equations. For example, attacks on EDON-R [Leu10, KNW09] and ESSENCE [NPRA+10] significantly benefit from this method, though the attacks are hard to scale and generalize. Dobbertin solved complex equations in the first attacks on MD4 and MD5 [Dob96, Dob98].

Reduced versions of (so far) strong primitives are also subject to manual solving. The collision attacks on reduced SHA-2 and are an example [NB08].

Dedicated tools. Some ad-hoc attacks can be generalized to automated tools, which are applicable to a large class of primitives. The triangulation algorithm [KBN09] transforms a system of equations in a way similar to the Gaussian elimination, and provides a number of solutions if the system is solvable. The tool is good for primitives with slow diffusion, and worse for bit-oriented transformations with data-dependent diffusion (like ARX). Similar tools were designed for stream ciphers [RS08, WB10].

The ARX-based designs have been analyzed with the concept of S-functions and automata theory (Section 3.3.3). These concepts also help to solve non-linear equations arising from the cryptanalysis of the designs [LT10].

Generic tools. Generic tools, like SAT-solvers or Gröbner basis algorithms, are always in the attention of cryptanalysts as methods that are applicable to any primitive. However, their universality is their weakness: there are very few attacks that benefited from generic tools. Among those who did, pretty many were outperformed later by dedicated methods. Some scholars say that generic tools are useful for small subroutines which have to be automated to save the time of an analyst, like search for a right pair for a short differential.

The following papers are typical examples of the application of generic tools to cryptanalysis [BKM10, Bul11, Sta10, BCN+10].

8.7 Unique attacks

This section is intentionally short. Here we briefly point out the attacks that has been applied only once, due to unusual (and apparently bad) design of a target. Though it is unlikely you will ever apply those methods, it is good to know them when designing a new ultra-lightweight, fast and secure primitive. After all, the most interesting examples are:

- Divide-and-conquer attack on Hummingbird [Saa11], whose four keys can be recovered sequentially.
- Cancellation attack on SHAvite-3 [GLM+10]. SHAvite has a generalized Feistel structure with two keyed nonlinear functions and four state variables at each step. The attacker fixes the key input to those functions so that outputs that apply to the same line of computation are equal: $F_k(x) = F_{k'}(x')$. As a result, the computation remains unchanged: $y \oplus F_k(x) \oplus F_{k'}(x') = y$.

Bibliography

[ABM^{+}09] Jean-Philippe Aumasson, Eric Brier, Willi Meier, María Naya-Plasencia, and Thomas Peyrin. Inside the hypercube. In *ACISP'09*, volume 5594 of *Lecture Notes in Computer Science*, pages 202–213. Springer, 2009.

[ABNP^{+}11] Mohamed Ahmed Abdelraheem, Céline Blondeau, María Naya-Plasencia, Marion Videau, and Erik Zenner. Cryptanalysis of ARMADILLO2. Available online at http://eprint.iacr.org/2011/160.pdf, 2011.

[ADIP09] Jean-Philippe Aumasson, Orr Dunkelman, Sebastiaan Indesteege, and Bart Preneel. Cryptanalysis of Dynamic SHA(2). In *Selected Areas in Cryptography'09*, volume 5867 of *Lecture Notes in Computer Science*, pages 415–432. Springer, 2009.

[ADMS09] Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube testers and key recovery attacks on reduced-round MD6 and Trivium. In *FSE'09*, volume 5665 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2009.

[AFK^{+}08] Jean-Philippe Aumasson, Simon Fischer, Shahram Khazaei, Willi Meier, and Christian Rechberger. New features of latin dances: Analysis of Salsa, ChaCha, and Rumba. In *FSE'08*, volume 5086 of *Lecture Notes in Computer Science*, pages 470–488. Springer, 2008.

[AGM^{+}09] Kazumaro Aoki, Jian Guo, Krystian Matusiewicz, Yu Sasaki, and Lei Wang. Preimages for step-reduced SHA-2. In *ASIACRYPT'09*, volume 5912 of *Lecture Notes in Computer Science*, pages 578–597. Springer, 2009.

[aJP02] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *ASIACRYPT'02*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer, 2002.

[ALZ11] Mohamed Ahmed Abdelraheem, Gregor Leander, and Erik Zenner. Differential cryptanalysis of round-reduced PRINTcipher: Computing roots of permutations. In *FSE'11*, volume 6733 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2011.

[AM09] Jean-Philippe Aumasson and Willi Meier. Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. NIST mailing list, 2009. availlable at http://www.131002.net/data/papers/AM09.pdf.

[AS08] Kazumaro Aoki and Yu Sasaki. Preimage attacks on one-block MD4, 63-step MD5 and more. In *Selected Areas in Cryptography'08*, volume 5381 of *Lecture Notes in Computer Science*, pages 103–119. Springer, 2008.

[AS09] Kazumaro Aoki and Yu Sasaki. Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1. In *CRYPTO'09*, volume 5677 of *Lecture Notes in Computer Science*, pages 70–89. Springer, 2009.

[BA07] Behran Bahrak and Mohammad Reza Aref. A novel impossible differential cryptanalysis of AES. In *Proceedings of the Western European Workshop on Research in Cryptology 2007 (WEWoRC'07)*, pages 152–156, 2007.

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Miss in the middle attacks on IDEA and Khufu. In *FSE'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 124–138. Springer, 1999.

[BBS06] Elad Barkan, Eli Biham, and Adi Shamir. Rigorous bounds on cryptanalytic time/memory tradeoffs. In *CRYPTO'06*, volume 4117 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2006.

[BC04] Eli Biham and Rafi Chen. Near-collisions of SHA-0. In *CRYPTO'04*, volume 3152 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2004.

[BCC11] Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-order differential properties of Keccak and Luffa. In *FSE'11*, volume 6733 of *Lecture Notes in Computer Science*, pages 252–269. Springer, 2011.

[BCCM$^+$09] Emmanuel Bresson, Anne Canteaut, Benoit Chevallier-Mames, Christophe Clavier, Thomas Fuhr, Aline Gouget, Thomas Icart, ois Misarsky Jean-Franc, Mari'a Naya-Plasencia, Pascal Paillier, Thomas Pornin, Jean-Rene' Reinhard, Céline Thuillet, and Marion Videau. Indifferentiability with distinguishers: Why Shabal does not require ideal ciphers. Cryptology ePrint Archive, Report 2009/199, available at http://eprint.iacr.org/2009/199.pdf, 2009.

[BCJ$^+$05] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and reduced SHA-1. In *EUROCRYPT'05*, volume 3494 of *Lecture Notes in Computer Science*, pages 36–57. Springer, 2005.

[BCN$^+$10] Gregory V. Bard, Nicolas Courtois, Jorge Nakahara, Pouyan Sepehrdad, and Bingsheng Zhang. Algebraic, AIDA/cube and side channel analysis of KATAN family of block ciphers. In *INDOCRYPT'10*, volume 6498 of *Lecture Notes in Computer Science*, pages 176–196. Springer, 2010.

[BCQ04] Alex Biryukov, Christophe De Cannière, and Michaël Quisquater. On multiple linear approximations. In *CRYPTO'04*, volume 3152 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2004.

[BDD$^+$10] Charles Bouillaguet, Patrick Derbez, Orr Dunkelman, Nathan Keller, Vincent Rijmen, and Pierre-Alain Fouque. Low data complexity attacks on aes. Available online at http://eprint.iacr.org/2010/633.pdf, 2010.

[BDK01] Eli Biham, Orr Dunkelman, and Nathan Keller. The rectangle attack - rectangling the Serpent. In *EUROCRYPT'01*, volume 2045 of *Lecture Notes in Computer Science*, pages 340–357. Springer, 2001.

[BDK02] Eli Biham, Orr Dunkelman, and Nathan Keller. New results on boomerang and rectangle attacks. In *FSE'02*, volume 2365 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2002.

[BDK05a] Eli Biham, Orr Dunkelman, and Nathan Keller. New combined attacks on block ciphers. In *FSE'05*, volume 3557 of *Lecture Notes in Computer Science*, pages 126–144. Springer, 2005.

[BDK05b] Eli Biham, Orr Dunkelman, and Nathan Keller. Related-key boomerang and rectangle attacks. In *EUROCRYPT'05*, volume 3494 of *Lecture Notes in Computer Science*, pages 507–525. Springer, 2005.

[BDK07] Eli Biham, Orr Dunkelman, and Nathan Keller. A new attack on 6-round IDEA. In *FSE'07*, volume 4593 of *Lecture Notes in Computer Science*, pages 211–224. Springer, 2007.

[BDK$^+$10] Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich, and Adi Shamir. Key recovery attacks of practical complexity on AES-256 variants with up to 10 rounds. In *EUROCRYPT'10*, volume 6110 of *Lecture Notes in Computer Science*, pages 299–319. Springer, 2010.

[BDLF09] Charles Bouillaguet, Orr Dunkelman, Gaëtan Leurent, and Pierre-Alain Fouque. Attacks on hash functions based on generalized Feistel — application to reduced-round Lesamnta and SHAvite-3_{512}. Cryptology ePrint Archive, Report 2009/634, available at http://eprint.iacr.org/2009/634.pdf, 2009.

[BDLF10] Charles Bouillaguet, Orr Dunkelman, Gae"tan Leurent, and Pierre-Alain Fouque. Another look at complementation properties. In *FSE'10*, volume 6147 of *Lecture Notes in Computer Science*, pages 347–364. Springer, 2010.

[BDPA06] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. RadioGatun, a belt-and-mill hash function. *NIST Cryptographic Hash Workshop, available at http://radiogatun.noekeon.org/*, 2006.

[BDPA07] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Sponge functions, available at http://sponge.noekeon.org/, 2007.

[BDPA09] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak sponge function family main document. Submission to NIST (updated), 2009. Version 1.2.

[BFL10] Charles Bouillaguet, Pierre-Alain Fouque, and Gaëtan Leurent. Security analysis of SIMD. In *Selected Areas in Cryptography'10*, volume 6544 of *Lecture Notes in Computer Science*, pages 351–368. Springer, 2010.

[BG09] Céline Blondeau and Benoît Gérard. On the data complexity of statistical attacks against block ciphers (full version). In Alexander Kholosha, Eirik Rosnes, and Matthew G. Parker, editors, *Workshop on Coding and Cryptography - WCC 2009*, pages 469–488, 2009.

[BGT11] Céline Blondeau, Benoît Gérard, and Jean-Pierre Tillich. Accurate estimates of the data complexity and success probability for various cryptanalyses. *DCC special issue on Coding and Cryptography*, 59(1-3):3–34, 2011.

[Bih94] Eli Biham. New types of cryptanalytic attacks using related keys. *J. Cryptology*, 7(4):229–246, 1994.

[Bir04] Alex Biryukov. The boomerang attack on 5 and 6-round reduced AES. In *AES Conference'04*, volume 3373 of *Lecture Notes in Computer Science*, pages 11–15. Springer, 2004.

[BK98a] Alex Biryukov and Eyal Kushilevitz. From differential cryptanalysis to ciphertext-only attacks. In *CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 72–88. Springer, 1998.

[BK98b] Alex Biryukov and Eyal Kushilevitz. Improved cryptanalysis of RC5. In *EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 85–99. Springer, 1998.

[BK03] Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In *EUROCRYPT'03*, volume 2656 of *Lecture Notes in Computer Science*, pages 491–506. Springer, 2003.

[BK09] Alex Biryukov and Dmitry Khovratovich. Related-key cryptanalysis of the full AES-192 and AES-256. In *ASIACRYPT'09*, volume 5912 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2009.

[BKLT11] Julia Borghoff, Lars Ramkilde Knudsen, Gregor Leander, and Søren Steffen Thomsen. Cryptanalysis of PRESENT-like ciphers with secret S-boxes. In *FSE'11*, volume 6733 of *Lecture Notes in Computer Science*, pages 270–289. Springer, 2011.

[BKM10] Julia Borghoff, Lars R. Knudsen, and Krystian Matusiewicz. Hill climbing algorithms and Trivium. In *Selected Areas in Cryptography'10*, volume 6544 of *Lecture Notes in Computer Science*, pages 57–73. Springer, 2010.

[BKMP09] Eric Brier, Shahram Khazaei, Willi Meier, and Thomas Peyrin. Linearization framework for collision attacks: Application to CubeHash and MD6. In *ASIACRYPT'09*, volume 5912 of *Lecture Notes in Computer Science*, pages 560–577. Springer, 2009.

[BKN09] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolić. Distinguisher and related-key attack on the full AES-256. In *CRYPTO'09*, volume 5677 of *Lecture Notes in Computer Science*, pages 231–249. Springer, 2009.

[BLP03] Alex Biryukov, Joseph Lano, and Bart Preneel. Cryptanalysis of the alleged SecurID hash function. In *Selected Areas in Cryptography'03*, volume 3006 of *Lecture Notes in Computer Science*, pages 130–144. Springer, 2003.

[BM97] Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *EUROCRYPT'97*, pages 163–192, 1997.

[BMS05] Alex Biryukov, Sourav Mukhopadhyay, and Palash Sarkar. Improved time-memory trade-offs with multiple data. In *Selected Areas in Cryptography'05*, volume 3897 of *Lecture Notes in Computer Science*, pages 110–127. Springer, 2005.

[BN10] Alex Biryukov and Ivica Nikolic. Automatic search for related-key differential characteristics in byte-oriented block ciphers: Application to AES, Camellia, Khazad and others. In *EUROCRYPT'10*, volume 6110 of *Lecture Notes in Computer Science*, pages 322–344. Springer, 2010.

[BN11] Alex Biryukov and Ivica Nikolic. Search for related-key differential characteristics in DES-like ciphers. In *FSE'11*, volume 6733 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2011.

[Bog08] Andrey Bogdanov. Linear slide attacks on the KeeLoq block cipher. In *Inscrypt'07*, volume 4990 of *Lecture Notes in Computer Science*. Springer, 2008.

[BPSZ10] Alex Biryukov, Deike Priemuth-Schmid, and Bin Zhang. Multiset collision attacks on reduced-round SNOW 3G and SNOW 3g $^{(+)}$. In *ACNS'10*, volume 6123 of *Lecture Notes in Computer Science*, pages 139–153, 2010.

[BR10] Andrey Bogdanov and Christian Rechberger. A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher KTANTAN. In *Selected Areas in Cryptography'10*, volume 6544 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2010.

[BR11] Andrey Bogdanov and Vincent Rijmen. Zero-correlation linear cryptanalysis of block ciphers. Available online at http://eprint.iacr.org/2011/123.pdf, 2011.

[BRS02] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In *CRYPTO'02*, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2002.

[BS92] Eli Biham and Adi Shamir. Differential cryptanalysis of the full 16-round DES. In *CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 487–496. Springer, 1992.

[BS01] Alex Biryukov and Adi Shamir. Structural cryptanalysis of SASAS. In *EUROCRYPT'01*, volume 2045 of *Lecture Notes in Computer Science*, pages 394–405. Springer, 2001.

[Bul11] Stanislav Bulygin. Algebraic cryptanalysis of the round-reduced and side channel analysis of the full PRINTCipher-48. Available online at http://eprint.iacr.org/2011/287.pdf, 2011.

[BW99] Alex Biryukov and David Wagner. Slide attacks. In *FSE'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 1999.

[BW00] Alex Biryukov and David Wagner. Advanced slide attacks. In *EUROCRYPT'00*, volume 1807 of *Lecture Notes in Computer Science*, pages 589–606. Springer, 2000.

[CBW08] Nicolas Courtois, Gregory V. Bard, and David Wagner. Algebraic and slide attacks on KeeLoq. In *FSE'08*, volume 5086 of *Lecture Notes in Computer Science*, pages 97–115. Springer, 2008.

[CC98] Anne Canteaut and Florent Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, 1998.

[Cho10] Joo Yeon Cho. Linear cryptanalysis of reduced-round PRESENT. In *CT-RSA'10*, volume 5985 of *Lecture Notes in Computer Science*, pages 302–317. Springer, 2010.

[CJ98] Florent Chabaud and Antoine Joux. Differential collisions in SHA-0. In *CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*. Springer, 1998.

[CKPS00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *EUROCRYPT'00*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer, 2000.

[CLO07] David A. Cox, John B. Little, and Don O'Shea. *Ideals, Varieties, and Algorithms.* Springer, 2007.

[CMR07] Christophe De Cannière, Florian Mendel, and Christian Rechberger. Collisions for 70-step SHA-1: On the full cost of collision search. In *Selected Areas in Cryptography'07*, volume 4876 of *Lecture Notes in Computer Science*, pages 56–73. Springer, 2007.

[CR06] Christophe De Cannière and Christian Rechberger. Finding SHA-1 characteristics: General results and applications. In *ASIACRYPT'06*, volume 4284 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006.

[CR08] Christophe De Cannière and Christian Rechberger. Preimages for reduced SHA-0 and SHA-1. In *CRYPTO'08*, volume 5157 of *Lecture Notes in Computer Science*, pages 179–202. Springer, 2008.

[DA07] Joan Daemen and Gilles Van Assche. Producing collisions for Panama, instantaneously. In *FSE'07*, volume 4593 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2007.

[Dam89] Ivan Damgård. A design principle for hash functions. In *CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1989.

[Dau05] Magnus Daum. *Cryptanalysis of Hash Functions of the MD4-Family.* PhD thesis, Ruhr-Universitat Bochum, 2005.

[DG08] Blandine Debraize and Louis Goubin. Guess-and-determine algebraic attack on the self-shrinking generator. In *FSE'08*, volume 5086 of *Lecture Notes in Computer Science*, pages 235–252. Springer, 2008.

[DGP$^+$11] Itai Dinur, Tim Gneysu, Christof Paar, Adi Shamir, and Ralf Zimmermann. An experimentally verified attack on full grain-128 using dedicated reconfigurable hardware. Available online at http://eprint.iacr.org/2011/282.pdf, 2011.

[DGV93] Joan Daemen, René Govaerts, and Joos Vandewalle. Weak keys for IDEA. In *CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 224–231. Springer, 1993.

[DIK08] Orr Dunkelman, Sebastiaan Indesteege, and Nathan Keller. A differential-linear attack on 12-round Serpent. In *INDOCRYPT'08*, volume 5365 of *Lecture Notes in Computer Science*, pages 308–321. Springer, 2008.

[DK09] Orr Dunkelman and Nathan Keller. Cryptanalysis of CTC2. In *CT-RSA'09*, volume 5473 of *Lecture Notes in Computer Science*, pages 226–239. Springer, 2009.

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher Square. In *FSE'97*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 1997.

[DKS10a] Orr Dunkelman, Nathan Keller, and Adi Shamir. Improved single-key attacks on 8-round AES-192 and AES-256. In *ASIACRYPT'10*, volume 6477 of *Lecture Notes in Computer Science*, pages 158–176. Springer, 2010.

[DKS10b] Orr Dunkelman, Nathan Keller, and Adi Shamir. A practical-time related-key attack on the KASUMI cryptosystem used in GSM and 3G telephony. In *CRYPTO'10*, volume 6223 of *Lecture Notes in Computer Science*, pages 393–410. Springer, 2010.

[DKS11] Orr Dunkelman, Nathan Keller, and Adi Shamir. ALRED blues: New attacks on AES-based MAC's. Available online at http://eprint.iacr.org/2011/095.pdf, 2011.

[DL05] Magnus Daum and Stefan Lucks. Hash collisions (the poisoned message attack). Technical report, Eurocrypt 2005 Rump Session, 2005.

[DL11] Ming Duan and Xuajia Lai. Improved zero-sum distinguisher for full round Keccak-f permutation. Available online at http://eprint.iacr.org/2011/023.pdf, 2011.

[DLP$^+$09] Joan Daemen, Mario Lamberger, Norbert Pramstaller, Vincent Rijmen, and Frederik Vercauteren. Computational aspects of the expected differential probability of 4-round AES and AES-like ciphers. *Computing*, 85(1-2):85–104, 2009.

[Dob96] Hans Dobbertin. The status of MD5 after a recent attack. *CryptoBytes*, 2(2):1–6, 1996. available at ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto2n2.pdf.

[Dob98] Hans Dobbertin. Cryptanalysis of MD4. *J. Cryptology*, 11(4):253–271, 1998.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael. AES — the Advanced Encryption Standard*. Springer, 2002.

[DR05] Joan Daemen and Vincent Rijmen. Probability distributions of correlation and differentials in block ciphers, 2005. Available at http://eprint.iacr.org/2005/212.pdf.

[DS08] Hüseyin Demirci and Ali Aydin Selçuk. A meet-in-the-middle attack on 8-round AES. In *FSE'08*, volume 5086 of *Lecture Notes in Computer Science*, pages 116–126. Springer, 2008.

[DS09] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In *EUROCRYPT'09*, volume 5479 of *Lecture Notes in Computer Science*, pages 278–299. Springer, 2009.

[DS11a] Itai Dinur and Adi Shamir. Breaking Grain-128 with dynamic cube attacks. In *FSE'11*, volume 6733 of *Lecture Notes in Computer Science*, pages 167–187. Springer, 2011.

[DS11b] Itai Dinur and Adi Shamir. An improved algebraic attack on Hamsi-256. In *FSE'11*, volume 6733 of *Lecture Notes in Computer Science*, pages 88–106. Springer, 2011.

[DSP07] Orr Dunkelman, Gautham Sekar, and Bart Preneel. Improved meet-in-the-middle attacks on reduced-round DES. In *INDOCRYPT'07*, volume 4859 of *Lecture Notes in Computer Science*, pages 86–100. Springer, 2007.

[DST04] Hüseyin Demirci, Ali Aydin Selçuk, and Erkan Türe. A new meet-in-the-middle attack on the IDEA block cipher. In *SAC'03*, volume 3006 of *Lecture Notes in Computer Science*, pages 117–129. Springer, 2004.

[DWS10] Zhenli Dai, Meiqin Wang, and Yue Sun. Effect of the dependent paths in linear hull. available at `http://eprint.iacr.org/2010/325.pdf`, 2010.

[FKL$^+$00] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David Wagner, and Doug Whiting. Improved cryptanalysis of Rijndael. In *FSE'00*, volume 1978 of *Lecture Notes in Computer Science*, pages 213–230. Springer, 2000.

[FLN07] Pierre-Alain Fouque, Gaëtan Leurent, and Phong Q. Nguyen. Full key-recovery attacks on HMAC/NMAC-MD4 and NMAC-MD5. In *CRYPTO'07*, volume 4622 of *Lecture Notes in Computer Science*, pages 13–30. Springer, 2007.

[FLS$^+$08] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The Skein hash function family. Submission to NIST (Round 1), available at `http://www.skein-hash.info/sites/default/files/skein.pdf`, 2008.

[FLZ$^+$10] Xiutao Feng, Jun Liu, Zhaocun Zhou, Chuankun Wu, and Dengguo Feng. A byte-based guess and determine attack on SOSEMANUK. In *ASIACRYPT'10*, volume 6477 of *Lecture Notes in Computer Science*, pages 146–157. Springer, 2010.

[FP09] Thomas Fuhr and Thomas Peyrin. Cryptanalysis of RadioGatún. In *FSE'09*, volume 5665 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 2009.

[Fuh10] Thomas Fuhr. Finding second preimages of short messages for Hamsi-256. In *ASIACRYPT'10*, volume 6477 of *Lecture Notes in Computer Science*, pages 20–37. Springer, 2010.

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GLM$^+$10] Praveen Gauravaram, Gaëtan Leurent, Florian Mendel, María Naya-Plasencia, Thomas Peyrin, Christian Rechberger, and Martin Schläffer. Cryptanalysis of the 10-round hash and full compression function of SHAvite-3-512. In *AFRICACRYPT'10*, volume 6055 of *Lecture Notes in Computer Science*, pages 419–436. Springer, 2010.

[GLP08] Michael Gorski, Stefan Lucks, and Thomas Peyrin. Slide attacks on a class of hash functions. In *ASIACRYPT'08*, volume 5350 of *Lecture Notes in Computer Science*, pages 143–160. Springer, 2008.

[GLRW10] Jian Guo, San Ling, Christian Rechberger, and Huaxiong Wang. Advanced meet-in-the-middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2. In *ASIACRYPT'10*, volume 6477 of *Lecture Notes in Computer Science*, pages 56–75. Springer, 2010.

[GM00] Henri Gilbert and Marine Minier. A collision attack on 7 rounds of Rijndael. In *AES Candidate Conference*, pages 230–241, 2000.

[GM08] Samuel Galice and Marine Minier. Improving integral attacks against Rijndael-256 up to 9 rounds. In *AFRICACRYPT'08*, volume 5023 of *Lecture Notes in Computer Science*. Springer, 2008.

[GM09] Jian Guo and Krystian Matusiewicz. Round-reduced near-collisions of BLAKE-32. Available online at http://www.jguo.org/docs/blake-col.pdf, 2009. Accepted for presentation at WEWoRC 2009.

[GMK+09] Jian Guo, Krystian Matusiewicz, Lars R. Knudsen, San Ling, and Huaxiong Wang. Practical pseudo-collisions for hash functions Arirang-224/384. Available online at http://ehash.iaik.tugraz.at/uploads/9/9a/Arirang-pseudo-sha3zoo.pdf, 2009.

[GP10] Henri Gilbert and Thomas Peyrin. Super-sbox cryptanalysis: Improved attacks for AES-like permutations. In *FSE'10*, volume 6147 of *Lecture Notes in Computer Science*, pages 365–383. Springer, 2010.

[Hel80] Martin Hellman. A cryptanalytic time-memory trade-off. *IEEE transactions on Information Theory*, 26:401–406, 1980.

[HM97] Carlo Harpes and James L. Massey. Partitioning cryptanalysis. In *FSE'97*, volume 1267 of *Lecture Notes in Computer Science*, pages 13–27. Springer, 1997.

[HN10] Miia Hermelin and Kaisa Nyberg. Dependent linear approximations: The algorithm of Biryukov and others revisited. In *CT-RSA'10*, volume 5985 of *Lecture Notes in Computer Science*, pages 318–333. Springer, 2010.

[HN11] Miia Hermelin and Kaisa Nyberg. Linear cryptanalysis using multiple linear approximations. Available online at http://eprint.iacr.org/2011/093.pdf, 2011.

[HQ01] Yeping He and Sihan Qing. Square attack on reduced Camellia cipher. In *ICICS'01*, volume 2229 of *Lecture Notes in Computer Science*, pages 238–245. Springer, 2001.

[HR02] Philip Hawkes and Gregory G. Rose. Guess-and-determine attacks on SNOW. In *Selected Areas in Cryptography'02*, volume 2595 of *Lecture Notes in Computer Science*, pages 37–46. Springer, 2002.

[HSK02] Yasuo Hatano, Hiroki Sekine, and Toshinobu Kaneko. Higher order differential attack of Camellia (ii). In *Selected Areas in Cryptography'02*, volume 2595 of *Lecture Notes in Computer Science*, pages 129–146. Springer, 2002.

[IKD$^+$08] Sebastiaan Indesteege, Nathan Keller, Orr Dunkelman, Eli Biham, and Bart Preneel. A practical attack on KeeLoq. In *EUROCRYPT'08*, volume 4965 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.

[IMPS09] Sebastiaan Indesteege, Florian Mendel, Bart Preneel, and Martin Schläffer. Practical collisions for SHAMATA-256. In *Selected Areas in Cryptography'09*, volume 5867 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2009.

[IP08] Sebastiaan Indesteege and Bart Preneel. Collisions for RC4-Hash. In *ISC'08*, volume 5222 of *Lecture Notes in Computer Science*, pages 355–366. Springer, 2008.

[Iso11] Takanori Isobe. A single-key attack on the full GOST block cipher. In *FSE'11*, volume 6733 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2011.

[IW09] Kota Ideguchi and Dai Watanabe. Second preimage attack on SHAMATA-512. In *INDOCRYPT'09*, volume 5922 of *Lecture Notes in Computer Science*, pages 169–181. Springer, 2009.

[JdFP05] Jorge Nakahara Jr., Daniel Santana de Freitas, and Raphael Chung-Wei Phan. New multiset attacks on Rijndael with large blocks. In *Mycrypt'05*, volume 3715 of *Lecture Notes in Computer Science*, pages 277–295. Springer, 2005.

[JP07] Antoine Joux and Thomas Peyrin. Hash functions and the (amplified) boomerang attack. In *CRYPTO'07*, volume 4622 of *Lecture Notes in Computer Science*, pages 244–263. Springer, 2007.

[JR94] Burton S. Kaliski Jr. and Matthew J. B. Robshaw. Linear cryptanalysis using multiple approximations. In *CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 26–39. Springer, 1994.

[Jun05] Pascal Junod. *Statistical Cryptanalysis of Block Ciphers.* PhD thesis, Ecole Polytechnique Federale de Lausanne, 2005.

[Kar08] Orhun Kara. Reflection cryptanalysis of some ciphers. In *INDOCRYPT'08*, volume 5365 of *Lecture Notes in Computer Science*, pages 294–307. Springer, 2008.

[KBN09] Dmitry Khovratiovich, Alex Biryukov, and Ivica Nikolić. Speeding up collision search for byte-oriented hash functions. In *CT-RSA'09*, volume 5473 of *Lecture Notes in Computer Science*, pages 164–181. Springer, 2009.

[KHL$^+$04] Youngdai Ko, Seokhie Hong, Wonil Lee, Sangjin Lee, and Ju-Sung Kang. Related key differential attacks on 27 rounds of XTEA and full-round GOST. In *FSE'04*, volume 3017 of *Lecture Notes in Computer Science*, pages 299–316. Springer, 2004.

[Kho08] Dmitry Khovratovich. Two attacks on RadioGatún. In *INDOCRYPT'08*, volume 5365 of *Lecture Notes in Computer Science*, pages 53–66. Springer, 2008.

[Kho09a] Dmitry Khovratovich. Cryptanalysis of hash functions with structures. In *Selected Areas in Cryptography'09*, volume 5867 of *Lecture Notes in Computer Science*, pages 108–125. Springer, 2009.

[Kho09b] Dmitry Khovratovich. Nonrandomness of the 33-round MD6. Technical report, FSE 2009 Rump Session, 2009.

[KHP07] Jongsung Kim, Seokhie Hong, and Bart Preneel. Related-key rectangle attacks on reduced AES-192 and AES-256. In *FSE'07*, volume 4593 of *Lecture Notes in Computer Science*, pages 225–241. Springer, 2007.

[KK06] John Kelsey and Tadayoshi Kohno. Herding hash functions and the Nostradamus attack. In *EUROCRYPT'06*, volume 4004 of *Lecture Notes in Computer Science*, pages 183–200. Springer, 2006.

[KKMS10] Shahram Khazaei, Simon Knellwolf, Willi Meier, and Deian Stefan. Improved linear differential attacks on CubeHash. In *AFRICACRYPT'10*, volume 6055 of *Lecture Notes in Computer Science*, pages 407–418. Springer, 2010.

[KKS00] John Kelsey, Tadayoshi Kohno, and Bruce Schneier. Amplified boomerang attacks against reduced-round MARS and Serpent. In *FSE'00*, volume 1978 of *Lecture Notes in Computer Science*, pages 75–93. Springer, 2000.

[KL06] John Kelsey and Stefan Lucks. Collisions and near-collisions for reduced-round Tiger. In *FSE'06*, volume 4047 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 2006.

[Kli06] Vlastimil Klima. Tunnels in hash functions: MD5 collisions within a minute. Available at http://eprint.iacr.org/2006/105.pdf, 2006.

[KMNP10] Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional differential cryptanalysis of NLFSR-based cryptosystems. In *ASIACRYPT'10*, volume 6477 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2010.

[KN10] Dmitry Khovratovich and Ivica Nikolić. Rotational cryptanalysis of ARX. In *FSE'10*, volume 6147 of *Lecture Notes in Computer Science*, pages 333–346. Springer, 2010.

[KNPRS10] Dmitry Khovratovich, María Naya-Plasencia, Andrea Röck, and Martin Schläffer. Cryptanalysis of Luffa v2 components. In *Selected Areas in Cryptography'10*, volume 6544 of *Lecture Notes in Computer Science*, pages 388–409. Springer, 2010.

[KNR10] Dmitry Khovratovich, Ivica Nikolic, and Christian Rechberger. Rotational rebound attacks on reduced Skein. In *ASIACRYPT'10*, volume 6477 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2010.

[Knu94] Lars R. Knudsen. Truncated and higher order differentials. In *FSE'94*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1994.

[KNW09] Dmitry Khovratovich, Ivica Nikolić, and Ralf-Philipp Weinmann. Meet-in-the-middle attacks on SHA-3 candidates. In *FSE'09*, volume 5665 of *Lecture Notes in Computer Science*, pages 228–245. Springer, 2009.

[KR96] Lars R. Knudsen and Matthew J. B. Robshaw. Non-linear approximations in linear cryptoanalysis. In *EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 224–236. Springer, 1996.

[KR99] Lars R. Knudsen and Vincent Rijmen. On the decorrelated fast cipher (DFC) and its theory. In *FSE'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 1999.

[KR07] Lars R. Knudsen and Vincent Rijmen. Known-key distinguishers for some block ciphers. In *ASIACRYPT'07*, volume 4833 of *Lecture Notes in Computer Science*, pages 315–324. Springer, 2007.

[KR11] Dmitry Khovratovich and Christian Rechberger. A splice-and-cut cryptanalysis of the AES. Available online at http://eprint.iacr.org/2011/274.pdf, 2011.

[KRS11] Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. Bicliques for preimages: Attacks on Skein-512 and the SHA-2 family. Available online at http://eprint.iacr.org/2011/286.pdf, 2011.

[KRT07] Lars R. Knudsen, Christian Rechberger, and Søren S. Thomsen. The Grindahl hash functions. In *FSE'07*, volume 4593 of *Lecture Notes in Computer Science*, pages 39–57. Springer, 2007.

[KS05] John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In *EUROCRYPT'05*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer, 2005.

[KSW97] John Kelsey, Bruce Schneier, and David Wagner. Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA. In *ICICS'97*, volume 1334 of *Lecture Notes in Computer Science*, pages 233–246. Springer, 1997.

[KW02] Lars R. Knudsen and David Wagner. Integral cryptanalysis. In *FSE'02*, volume 2365 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2002.

[LCK$^+$08] Eunjin Lee, Donghoon Chang, Jongsung Kim, Jaechul Sung, and Seokhie Hong. Second preimage attack on 3-pass HAVAL and partial key-recovery attacks on HMAC/NMAC-3-pass HAVAL. In *FSE'08*, volume 5086 of *Lecture Notes in Computer Science*. Springer, 2008.

[LDKK08] Jiqiang Lu, Orr Dunkelman, Nathan Keller, and Jongsung Kim. New impossible differential attacks on AES. In *INDOCRYPT'08*, volume 5365 of *Lecture Notes in Computer Science*, pages 279–293. Springer, 2008.

[Lea11] Gregor Leander. On linear hulls, statistical saturation attacks, PRESENT and a cryptanalysis of PUFFIN. In *EUROCRYPT'11*, volume 6632 of *Lecture Notes in Computer Science*, pages 303–322. Springer, 2011.

[Leo88] Jeffrey S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354–1359, 1988.

[Leu08] Gaëtan Leurent. MD4 is not one-way. In *FSE'08*, volume 5086 of *Lecture Notes in Computer Science*, pages 412–428. Springer, 2008.

[Leu10] Gaëtan Leurent. Practical key recovery attack against secret-IV Edon-R. In *CT-RSA'10*, volume 5985 of *Lecture Notes in Computer Science*, pages 334–349. Springer, 2010.

[LH94] Susan K. Langford and Martin E. Hellman. Differential-linear cryptanalysis. In *CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 17–25. Springer, 1994.

[LM91] Xuejia Lai and James L. Massey. Markov ciphers and differential cryptanalysis. In *EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38. Springer, 1991.

[LM01] Helger Lipmaa and Shiho Moriai. Efficient algorithms for computing differential properties of addition. In *FSE'01*, volume 2355 of *Lecture Notes in Computer Science*, pages 336–350. Springer, 2001.

[LM11] Mario Lamberger and Florian Mendel. Higher-order differential attack on reduced SHA-256. Available online at http://eprint.iacr.org/2011/095.pdf, 2011.

[LMR$^+$09] Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. Rebound distinguishers: Results on the full Whirlpool compression function. In *ASIACRYPT'09*, volume 5912 of *Lecture Notes in Computer Science*, pages 126–143. Springer, 2009.

[LSL10] Ruilin Li, Bing Sun, and Chao Li. Impossible differential cryptanalysis of SPN ciphers. Available online at http://eprint.iacr.org/2010/307.pdf, 2010.

[LSLQ10] Ruilin Li, Bing Sun, Chao Li, and Longjiang Qu. Cryptanalysis of a generalized unbalanced feistel network structure. In *ACISP'10*, volume 6168 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2010.

[LT10] Gaëtan Leurent and Søren S. Thomsen. Practical partial-collisions on the compression function of BMW. Available online at http://www.di.ens.fr/~leurent/files/BMW_Distinguisher.pdf, 2010.

[LT11] Gaëtan Leurent and Søren S. Thomsen. Practical near-collisions on the compression function of BMW. In *FSE'11*, volume 6733 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2011.

[Mat93] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.

[MDRMH10] Hamid Mala, Mohammad Dakhilalian, Vincent Rijmen, and Mahmoud Modarres-Hashemi. Improved Impossible Differential Cryptanalysis of 7-Round AES-128. In *INDOCRYPT'10*, volume 6498 of *Lecture Notes in Computer Science*, pages 282–291. Springer, 2010.

[MDS10] Hamid Mala, Mohammad Dakhilalian, and Mohsen Shakiba. Cryptanalysis of block ciphers using almost-impossible differentials. Available online at http://eprint.iacr.org/2010/485.pdf, 2010.

[Mer89] Ralph C. Merkle. One way hash functions and DES. In *CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer, 1989.

[MG00] Marine Minier and Henri Gilbert. Stochastic cryptanalysis of Crypton. In *FSE'00*, volume 1978 of *Lecture Notes in Computer Science*, pages 121–133. Springer, 2000.

[MK08] Alexander Maximov and Dmitry Khovratovich. New state recovery attack on RC4. In *CRYPTO'08*, volume 5157 of *Lecture Notes in Computer Science*, pages 297–316. Springer, 2008.

[MN09] Florian Mendel and Tomislav Nad. A distinguisher for the compression function of SIMD-512. In *INDOCRYPT'09*, volume 5922 of *Lecture Notes in Computer Science*, pages 219–232. Springer, 2009.

[MNPN$^+$09] Krystian Matusiewicz, Maria Naya-Plasencia, Ivica Nikolić, Yu Sasaki, and Martin Schläffer. Rebound attack on the full LANE compression function. In *ASIACRYPT'09*, volume 5912 of *Lecture Notes in Computer Science*, pages 106–125. Springer, 2009.

[MNPP11] Marine Minier, María Naya-Plasencia, and Thomas Peyrin. Analysis of reduced-SHAvite-3-256 v2. In *FSE'11*, volume 6733 of *Lecture Notes in Computer Science*, pages 68–87. Springer, 2011.

[MP05] Krystian Matusiewicz and Josef Pieprzyk. Finding good differential patterns for attacks on SHA-1. In *WCC'05*, volume 3969 of *Lecture Notes in Computer Science*, pages 164–177. Springer, 2005.

[MPR$^+$08] Florian Mendel, Norbert Pramstaller, Christian Rechberger, Marcin Kontak, and Janusz Szmidt. Cryptanalysis of the GOST hash function. In *CRYPTO'08*, volume 5157 of *Lecture Notes in Computer Science*, pages 162–178. Springer, 2008.

[MPRS09] Florian Mendel, Thomas Peyrin, Christian Rechberger, and Martin Schläffer. Improved cryptanalysis of the reduced Grøstl compression function, ECHO permutation and AES block cipher. In *Selected Areas in Cryptography'09*, volume 5867 of *Lecture Notes in Computer Science*, pages 16–35. Springer, 2009.

[MRST09] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The rebound attack: Cryptanalysis of reduced Whirlpool and Grøstl. In *FSE'09*, volume 5665 of *Lecture Notes in Computer Science*, pages 260–276. Springer, 2009.

[MRST10] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Rebound attacks on the reduced Grøstl hash function. In *CT-RSA'10*, volume 5985 of *Lecture Notes in Computer Science*, pages 350–365. Springer, 2010.

[Mul04] Frédéric Muller. The MD2 hash function is not one-way. In *ASIACRYPT'04*, volume 3329 of *Lecture Notes in Computer Science*, pages 214–229. Springer, 2004.

[Mur09] Sean Murphy. The return of the boomerang, available at http://www.isg.rhul.ac.uk/~sean/Boomerang_Return.pdf. Technical report, 2009.

[MVCP10] Nicky Mouha, Vesselin Velichkov, Christophe De Cannière, and Bart Preneel. The differential analysis of S-functions. In *SAC'10*, volume 6544 of *Lecture Notes in Computer Science*, pages 36–56. Springer, 2010.

[MZ06] Ilya Mironov and Lintao Zhang. Applications of SAT solvers to cryptanalysis of hash functions. In *Theory and Applications of Satisfiability Testing - SAT 2006*, volume 4121 of *Lecture Notes in Computer Science*, pages 102–115. Springer, 2006.

[NB08] Ivica Nikolic and Alex Biryukov. Collisions for step-reduced SHA-256. In *FSE'08*, volume 5086 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2008.

[NP11] María Naya-Plasencia. How to improve rebound attacks. In *CRYPTO'11*, Lecture Notes in Computer Science, page ?? Springer, 2011. Available online at http://eprint.iacr.org/2010/607.pdf.

[NPRA$^+$10] María Naya-Plasencia, Andrea Röck, Jean-Philippe Aumasson, Yann Laigle-Chapuy, Gaëtan Leurent, Willi Meier, and Thomas Peyrin. Cryptanalysis of ESSENCE. In *FSE'10*, volume 6147 of *Lecture Notes in Computer Science*, pages 134–152. Springer, 2010.

[NPSS10] Ivica Nikolić, Josef Pieprzyk, Przemyslaw Sokolowski, and Ron Steinfeld. Rotational cryptanalysis of (modified) versions of BMW and SIMD. Available online at https://cryptolux.org/mediawiki/uploads/0/07/Rotational_distinguishers_(Nikolic,_Pieprzyk,_Sokolowski,_Steinfeld).pdf, 2010.

[NSKO05] Yusuke Naito, Yu Sasaki, Noboru Kunihiro, and Kazuo Ohta. Improved collision attack on MD4 with probability almost 1. In *ICISC'05*, volume 3935 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2005.

[NSS$^+$06] Yusuke Naito, Yu Sasaki, Takeshi Shimoyama, Jun Yajima, Noboru Kunihiro, and Kazuo Ohta. Improved collision search for SHA-0. In *ASIACRYPT'06*, volume 4284 of *Lecture Notes in Computer Science*, pages 21–36. Springer, 2006.

[NTW10] Karsten Nohl, Erik Tews, and Ralf-Philipp Weinmann. Cryptanalysis of the DECT standard cipher. In *FSE'10*, volume 6147 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2010.

[Oec03] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *CRYPTO'03*, volume 2729 of *Lecture Notes in Computer Science*, pages 617–630. Springer, 2003.

[Pey07] Thomas Peyrin. Cryptanalysis of Grindahl. In *ASIACRYPT'07*, volume 4833 of *Lecture Notes in Computer Science*, pages 551–567. Springer, 2007.

[Pey10] Thomas Peyrin. Improved differential attacks for ECHO and Grøstl. In *CRYPTO'10*, volume 6223 of *Lecture Notes in Computer Science*, pages 370–392. Springer, 2010.

[PRR05] Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Exploiting coding theory for collision attacks on SHA-1. In *IMA Int. Conf.*, volume 3796 of *Lecture Notes in Computer Science*, pages 78–95. Springer, 2005.

[Rab78] Michael O. Rabin. Digitalized signatures. *Foundations of Secure Computations*, pages 155–168, 1978.

[Rec09] Christian Rechberger. *Cryptanalysis of Hash Functions*. PhD thesis, Graz University of Technology, Graz, Austria, January 2009.

[Riv02] Ronald Rivest. MIT lecture notes: Voting, homomorphic encryption, 2002. available at http://web.mit.edu/6.857/OldStuff/Fall02/handouts/L15-voting.pdf.

[Riv08] Ronald L. Rivest. The MD6 hash function – a proposal to NIST for SHA-3. Submission to NIST, available at http://groups.csail.mit.edu/cis/md6/submitted-2008-10-27/Supporting_Documentation/md6_report.pdf, 2008.

[RN11] Andrea Rck and Kaisa Nyberg. Exploiting linear hull in Matsuis Algorithm 1 (extended version). Available online at http://eprint.iacr.org/2011/285.pdf, 2011.

[RRPV01] Vincent Rijmen, Bart Van Rompay, Bart Preneel, and Joos Vandewalle. Producing collisions for PANAMA. In *FSE'01*, volume 2355 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 2001.

[RS04] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *FSE'04*, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388. Springer, 2004.

[RS08] Håvard Raddum and Igor Semaev. Solving multiple right hand sides linear equations. *Des. Codes Cryptography*, 49(1-3):147–160, 2008.

[SA08a] Yu Sasaki and Kazumaro Aoki. Preimage attacks on 3, 4, and 5-pass HAVAL. In *ASIACRYPT'08*, volume 5350 of *Lecture Notes in Computer Science*, pages 253–271. Springer, 2008.

[SA08b] Yu Sasaki and Kazumaro Aoki. Preimage attacks on step-reduced MD5. In *ACISP'08*, volume 5107 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2008.

[SA09a] Yu Sasaki and Kazumaro Aoki. Finding preimages in full MD5 faster than exhaustive search. In *EUROCRYPT'09*, volume 5479 of *Lecture Notes in Computer Science*, pages 134–152. Springer, 2009.

[SA09b] Yu Sasaki and Kazumaro Aoki. Meet-in-the-middle preimage attacks on double-branch hash functions: Application to RIPEMD and others. In *ACISP'09*, volume 5594 of *Lecture Notes in Computer Science*, pages 214–231. Springer, 2009.

[Saa03] Markku-Juhani Olavi Saarinen. Cryptanalysis of block ciphers based on SHA-1 and MD5. In *FSE'03*, volume 2887 of *Lecture Notes in Computer Science*, pages 36–44. Springer, 2003.

[Saa07] Markku-Juhani Olavi Saarinen. A meet-in-the-middle collision attack against the new FORK-256. In *INDOCRYPT'07*, volume 4859 of *Lecture Notes in Computer Science*, pages 10–17. Springer, 2007.

[Saa11] Markku-Juhani O. Saarinen. Cryptanalysis of hummingbird-1. In *FSE'11*, volume 6733 of *Lecture Notes in Computer Science*, pages 328–341. Springer, 2011.

[Sch10] Martin Schläffer. Subspace distinguisher for 5/8 rounds of the ECHO-256 hash function. In *Selected Areas in Cryptography'10*, volume 6544 of *Lecture Notes in Computer Science*, pages 369–387. Springer, 2010.

[SK98] Takeshi Shimoyama and Toshinobu Kaneko. Quadratic relation of S-box and its application to the linear attack of full round DES. In *CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 200–211. Springer, 1998.

[SLdW07] Marc Stevens, Arjen K. Lenstra, and Benne de Weger. Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In *EUROCRYPT'07*, volume 4515 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2007.

[SLW$^+$10] Yu Sasaki, Yang Li, Lei Wang, Kazuo Sakiyama, and Kazuo Ohta. Non-full-active Super-Sbox analysis: Applications to ECHO and Grøstl. In *ASIACRYPT'10*, volume 6477 of *Lecture Notes in Computer Science*, pages 38–55. Springer, 2010.

[SPGQ06] François-Xavier Standaert, Gilles Piret, Neil Gershenfeld, and Jean-Jacques Quisquater. SEA: A scalable encryption algorithm for small embedded applications. In *CARDIS'06*, volume 3928 of *Lecture Notes in Computer Science*, pages 222–236. Springer, 2006.

[SSA$^+$09] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen K. Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In *CRYPTO'09*, volume 5677 of *Lecture Notes in Computer Science*, pages 55–69. Springer, 2009.

[Sta10] Paul Stankovski. Greedy distinguishers and nonrandomness detectors. In *INDOCRYPT'10*, volume 6498 of *Lecture Notes in Computer Science*, pages 210–226. Springer, 2010.

[Ste88] Jacques Stern. A method for finding codewords of small weight. In *Coding Theory and Applications*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer, 1988.

[Ste07] Marc Stevens. On collisions for MD5. Master's thesis, Eindhoven University of Technology, Eindhoven, Netherlands, 2007.

[SWOK07] Yu Sasaki, Lei Wang, Kazuo Ohta, and Noboru Kunihiro. New message difference for MD4. In *FSE'07*, volume 4593 of *Lecture Notes in Computer Science*, pages 329–348. Springer, 2007.

[SY11] Yu Sasaki and Kan Yasuda. Known-key distinguishers for 11-round feistel ciphers: Application to collision attacks on their hashing modes. In *FSE'11*, volume 6733 of *Lecture Notes in Computer Science*, pages 397–415. Springer, 2011.

[Tez10] Cihangir Tezcan. The improbable differential attack: Cryptanalysis of reduced round CLEFIA. In *INDOCRYPT'10*, volume 6498 of *Lecture Notes in Computer Science*, pages 197–209. Springer, 2010.

[Tho10] Søren S. Thomsen. Pseudo-cryptanalysis of the original Blue Midnight Wish. In *FSE'10*, volume 6147 of *Lecture Notes in Computer Science*, pages 304–317. Springer, 2010.

[TSSK08] Yukiyasu Tsunoo, Teruo Saito, Maki Shigeri, and Takeshi Kawabata. Higher order differential attacks on reduced-round MISTY1. In *ICISC'08*, volume 5461 of *Lecture Notes in Computer Science*, pages 415–431. Springer, 2008.

[TWP07] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. Breaking 104 bit WEP in less than 60 seconds. In *WISA*, volume 4867 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2007.

[Van10] Gilles VanAssche. A rotational distinguisher on shabal's keyed permutation and its impact on the security proofs. Available online at http://gva.noekeon.org/papers/ShabalRotation.pdf, 2010.

[Vau96] Serge Vaudenay. An experiment on DES statistical cryptanalysis. In *ACM Conference on Computer and Communications Security*, pages 139–147, 1996.

[Wag99] David Wagner. The boomerang attack. In *FSE'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 1999.

[Wag02] David Wagner. A generalized birthday problem. In *CRYPTO'02*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002.

[Wal03] Johan Wallén. *On the differential and linear properties of addition*. PhD thesis, Helsinki University of Technology, 2003.

[WB10] Kenneth Koon-Ho Wong and Gregory V. Bard. Improved algebraic cryptanalysis of QUAD, Bivium and Trivium via graph partitioning on equation systems. In *ACISP'10*, volume 6168 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 2010.

[WFW09] Shuang Wu, Dengguo Feng, and Wenling Wu. Practical rebound attack on 12-round Cheetah-256. In *ICISC'09*. Springer, to appear, 2009.

[WHYK10] Dai Watanabe, Yasuo Hatano, Tsuyoshi Yamada, and Toshinobu Kaneko. Higher order differential attack on step-reduced variants of Luffa v1. In *FSE'10*, volume 6147 of *Lecture Notes in Computer Science*, pages 270–285. Springer, 2010.

[WLH10] Yongzhuang Wei, Jiqiang Lu, and Yupu Hu. Meet-in-the-middle attack on 8 rounds of the AES block cipher under 192 key bits. Available online at http://eprint.iacr.org/2010/537.pdf, 2010.

[WLSL10] Yuechuan Wei, Ping Li, Bing Sun, and Chao Li. Impossible differential cryptanalysis on Feistel ciphers with P and PS round functions. In *ACNS'10*, volume 6123 of *Lecture Notes in Computer Science*, pages 105–122, 2010.

[WOK08] Lei Wang, Kazuo Ohta, and Noboru Kunihiro. New key-recovery attacks on HMAC/NMAC-MD4 and NMAC-MD5. In *EUROCRYPT'08*, volume 4965 of *Lecture Notes in Computer Science*, pages 237–253. Springer, 2008.

[WRG$^+$11] Lei Wei, Christian Rechberger, Jian Guo, Hongjun Wu, Huaxiong Wang, and San Ling. Improved meet-in-the-middle cryptanalysis of KTANTAN. Available online at http://eprint.iacr.org/2011/201.pdf, 2011.

[WSK$^+$11] Lei Wang, Yu Sasaki, Wataru Komatsubara, Kazuo Ohta, and Kazuo Sakiyama. (second) preimage attacks on step-reduced RIPEMD/RIPEMD-128 with a new local-collision approach. In *CT-RSA'11*, volume 6558 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 2011.

[WY05] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *EUROCRYPT'05*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In *CRYPTO'05*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.

[WZF07] Wenling Wu, Wentao Zhang, and Dengguo Feng. Impossible differential cryptanalysis of reduced-round ARIA and Camellia. *J. Comput. Sci. Technol.*, 22(3):449–456, 2007.

[Yuv79] Gideon Yuval. How to swindle Rabin. *Cryptologia*, 3:187–189, 1979.

[YWZW05] Hongbo Yu, Gaoli Wang, Guoyan Zhang, and Xiaoyun Wang. The second-preimage attack on MD4. In *CANS'05*, volume 3810 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.

[ZL10] Jinmin Zhong and Xuejia Lai. Improved preimage attack on one-block MD4. Available online at http://eprint.iacr.org/2010/583.pdf, 2010.

[ZRHD08] Muhammad Reza Z'aba, Håvard Raddum, Matthew Henricksen, and Ed Dawson. Bit-pattern based integral attack. In *FSE'08*, volume 5086 of *Lecture Notes in Computer Science*, pages 363–381. Springer, 2008.