Analysis of Modern Dedicated Cryptographic Hash Functions

Krystian Matusiewicz, M.Sc.

This thesis is presented for the degree of

Doctor of Philosophy

Centre for Advanced Computing, Algorithms and Cryptography

Department of Computing

Division of Information and Communication Sciences

Macquarie University

August 2007

Contents

Abstract

The topic of this thesis is the study of cryptographic hash functions, one of the most important classes of primitives used in modern cryptography. The main aim is the development of new techniques of cryptanalysis of dedicated cryptographic hash functions.

We start with reviewing basic theoretical foundations of hashing. We present different approaches to defining security properties more formally and after discussing then we review basic attacks on hash functions.

We recall Merkle-Damgård iterative construction and discuss security properties of iterated hash functions. Next, we focus on practical designs of dedicated hash functions. We attempt to present dedicated designs in a unified framework and discuss some of their properties.

The main contribution of this thesis is the development of new cryptanalytical techniques applicable to dedicated cryptographic hash functions, mainly from the SHA family. We analyse functions SHA-1 and SHA-256 and show how the problem of finding low weight codewords in linear codes is relevant to differential cryptanalysis of those designs. We show previously unknown differentials of low weight in the message expansion of SHA-1 and discuss the possibility of using them to attack the whole function. We investigate the architecture of SHA-256 and present two attacks on simplified variants of that function. The first one shows that SHA-256 stripped of its diffusion boxes can be easily attacked. The latter one presents an analysis of short variants of SHA-256 with modular additions replaced by XORs and discusses the potential and limitations of this very general approach.

Moving to alternative designs, we present an attack on a recently proposed

hash function FORK-256. Again, we use coding theory tools to exhibit promising differential paths and exploit a particular weakness of the step transformation that results in a very restricted propagation of differences through the step transformation.

Finally, we investigate a more theoretical problem how the possibility of generating hashes that share a common part influences the complexity of finding collisions.

We conclude with a brief outline of possible future research directions in the area of cryptographic hash functions.

Declaration

This thesis is submitted in fulfilment of the requirements of the degree of Doctor of Philosophy at Macquarie University and has not been submitted for a higher degree to any other university or institution. This thesis represents my original work and contributions. I certify that to the best of my knowledge, all sources and assistance received in the preparation of this thesis have been acknowledged.

Krystian Matusiewicz

Thesis related publications

- Scott Contini, Krystian Matusiewicz and Josef Pieprzyk, "Extending FORK-256 Attack to the Full Hash Function, *Proc. ICICS 2007*, December 12-15, Zhengzhou, China; LNCS 4861, Springer, 2007"

- Scott Contini, Ron Steinfeld, Josef Pieprzyk and Krystian Matusiewicz, "A Critical Look at Cryptographic Hashing Literature", *ECRYPT Hash Workshop'07*, May 24 - 25, 2007, Barcelona, Spain

- Krystian Matusiewicz, Thomas Peyrin, Olivier Billet, Scott Contini and Josef Pieprzyk, "Cryptanalysis of FORK-256", *Proc. Fast Software Encryption 2007*, March 26-28, 2007, Luxembourg; LNCS 4593, Springer, 2007
  Extended version available as

  – Krystian Matusiewicz, Scott Contini and Josef Pieprzyk, "Weaknesses of the FORK-256 compression function", *IACR Cryptology e-print Archive, report 2006/317*

- Krystian Matusiewicz, Josef Pieprzyk, Norbert Pramstaller, Christian Rechberger, Vincent Rijmen, "Analysis of simplified variants of SHA-256", *Proc. Western European Workshop on Research in Cryptology, WEWoRC 2005*, July 5-7, 2005, Leuven, Belgium; Lecture Notes in Informatics, P-74, Gesellschaft für Informatik, 2005

- Krystian Matusiewicz, Josef Pieprzyk, "Finding good differential patterns for attacks on SHA-1", *Proc. International Workshop on Coding and Cryptography, WCC 2005*, March 14-18, 2005, Bergen, Norway; LNCS 3969, Springer, 2006

Acknowledgements

I would like to express my gratitude to my supervisor, Prof. Josef Pieprzyk for his help and guidance. During that time, he was not only a supervisor but also a mentor and a good friend. I am also grateful to my co-supervisor, Prof. Huaxiong Wang for his continuous support and interest in my research.

I would like to express my big thanks to Dr. Scott Contini for his invaluable help, many discussions and the inspiring example of a passionate researcher.

At the end of this research journey, I would also like to thank my fellow traveller, Christophe Tartary with whom I shared not only the office but also many interests and all the ups and downs of the PhD candidature.

I am happy I can acknowledge here the role of my parents, Grażyna and Janusz. Their love, care and the example of wisdom and integrity gave me a solid foundation to build my life on.

Finally, I cannot express enough my admiration and gratitude to my wonderful wife Sylwia, who supported me during that time in every possible way.

1

Introduction

A hash function is a function that maps strings of arbitrary length to strings of fixed length. This means that whether the input data is just a few words or the whole video file a few gigabytes long, the output of the function is always of the same length. The simplest example of a hash function is the modulo operation. When we represent input data as a number (any data can be represented as a string of bits and it can be seen as a binary representation of a number) and take the remainder of the division by a fixed value we always end up with a non-negative number smaller than that value.

There are many applications in which different kinds of hash functions are used, ranging from data structures such as hash tables [94, Section 6.4] through pattern-matching algorithms like [86] to checksum algorithms that help detecting accidental errors in data. They all rely on the fundamental property that *most of the times different input values yield different output values*, so the output of the hash function can be treated as a kind of a "fingerprint" of the input data that somehow identifies it. We can use this fingerprint to decide where to put

the data into the hash table, whether the particular substring is present in the text or if the data has been transmitted correctly.

However, this does not mean that with a bit of skills and patience one cannot find different input data that map to the same output for such functions. For simple hash functions such as the modulo operation it is trivial: it is enough to add a multiple of the modulus to get another number that yields the same output. The ability to construct different messages that result in the same output value may have serious consequences when exploited by a malicious person. One of the vivid examples is a denial-of-service attack presented in [41] that uses the ability to construct many messages with identical values of the hash function to degenerate hash tables used by the intrusion detection system and thus slow down the attacked application to the point where it cannot cope with incoming data anymore. Another real-life example is an attack on FastTrack peer-to-peer network. The FastTrack protocol [1], used by such popular file sharing programs as Kazaa and iMesh, uses a hash function called UUHash to identify files stored in nodes. This function is very weak as in the attempt to be maximally efficient it does not process the whole file but only some parts of it. It is trivial to find collisions for such a construction and this was used to "poison" the file sharing network with bogus files containing rubbish but with hashes still matching the original content. When such a fake was used when downloading a file, the resulting file is of course unusable.

Clearly, if we want to maintain this desired ability of a hash function to produce almost unique fingerprints of data even in the presence of a malicious adversary a stronger construction is required. This motivates the design of cryptographic hash functions.

Cryptographic hash functions Informally speaking, a cryptographic hash function is a hash function that can be computed efficiently and has three additional security properties, illustrated in Fig. 1.1: it should be preimage-resistant, second-preimage resistant and collision-resistant. We will investigate these notions in details in Chapter 2, here we want to convey only the basic intuition.

A function is preimage-resistant, if having the output value of the function, it is computationally infeasible to find any input that maps to that output. This

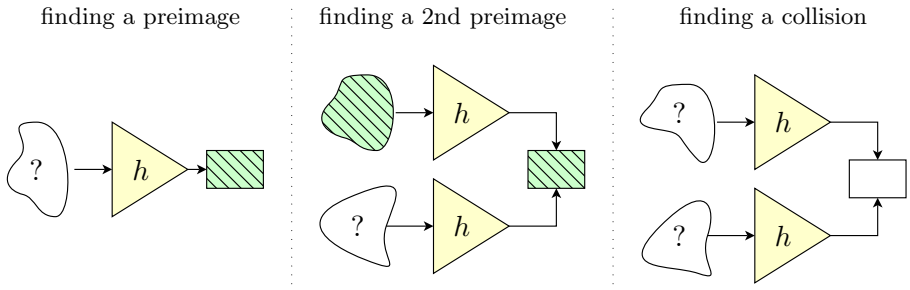


Figure 1.1: Illustration of basic security properties of a cryptographic hash function h. Dashed values are given as inputs and the aim is to find values marked with '?'.

property is also called one-wayness as one can easily compute the function in one way but cannot go back in the other direction.

A function is second preimage-resistant, if given the input value and the corresponding output it is computationally infeasible to find another distinct input that hashes to the same output.

Finally, a function is called collision-resistant, if it is infeasible to find a pair of different inputs that map to the identical value.

Quite often input data is called a message and outputs of cryptographic hash functions are called digests or just hashes.

We can interpret the preimage resistance property as inability to learn about the contents of the input data from its digest. One can compare it to the hardness of learning about the colour of eyes or hair of people based on the shape of their fingerprints.

The collision resistance means that digests are almost unique for each given message. If the message is changed, almost always the hash changes as well. The word almost is significant, because when we map larger domains to smaller ranges, collisions necessarily exist, but for properly designed cryptographic hash functions with digests of sufficient length the probability that one can stumble upon two different messages with identical hashes is so small that it can be disregarded in all practical applications.

Applications of cryptographic hash functions A function satisfying all the properties mentioned above is a powerful and versatile tool and can be used

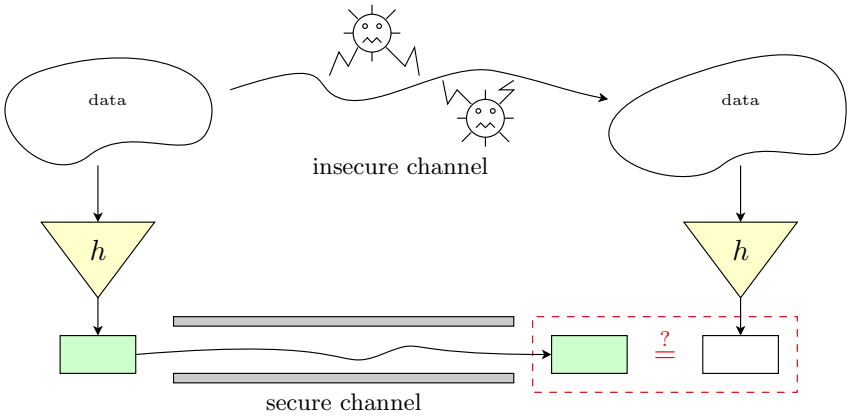


Figure 1.2: Comparing the digest of data sent over insecure communication channel with securely obtained original digest allows to verify integrity of the data.

to achieve a variety of security goals. We briefly recall here a couple of most common applications.

One of the prominent applications is information authentication. It enables the verification of the integrity of data sent over an insecure communication channel. The situation is presented in Fig. 1.2. Before sending the data its digest is computed by the means of a cryptographic hash function. The digest is then sent over a secure channel to the recipient who after receiving the original digest computes the hash of the received data and compares both hash values. If they are different, the information has been modified somehow on its way over the insecure channel. On the other hand, if the digests are identical, with overwhelming probability the message has not been altered, provided that the cryptographic hash function is secure. Indeed, if it is possible to find second preimages for the hash function in use, the adversary can manipulate the message on its way and still be able to come up with modification that hashes to the original value.

Another widespread application of cryptographic hash functions are digital signatures with appendix. Instead of using the signature algorithm to sign the original data directly, a cryptographic hash function is used to compute the digest of the message first and then, rather than the data, the digest is signed, cf. Fig. 1.3. During verification phase, the digest of data to be verified is computed and is used in the signature verification algorithm. The big advantage of this approach is its increased efficiency of signing long messages. Signature algo-

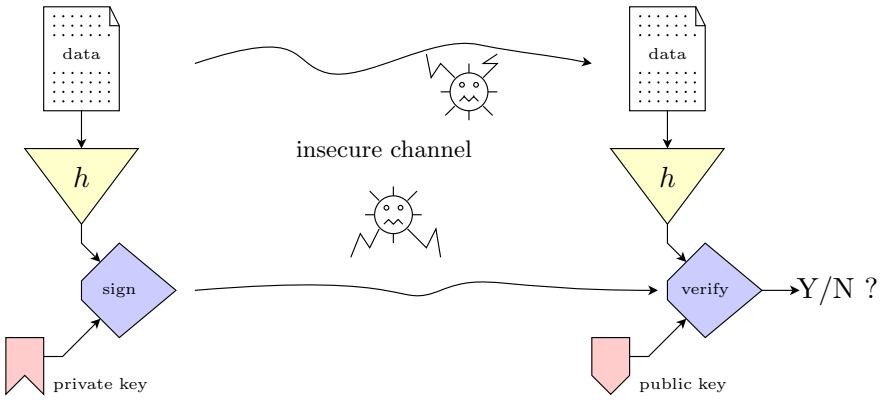


Figure 1.3: Digital signature schemes with appendix use cryptographic hash function h to obtain the digest of the data which is later signed.

rithms are much slower than hash functions and signing long messages directly would take a very long time. By computing cryptographic digest of the data first it is possible to avoid this costly computation. There are also some other, more technical advantages of using cryptographic hash functions with digital signatures when the hash function is used to destroy specific mathematical properties of the signature scheme. It is not hard to see that in order to retain the security of the digital signature the hash function must be collision resistant. Otherwise the signer could find different messages that yield the same digest and thus two different documents would have the same signature. This could lead to ambiguity as to which document was actually committed to by the signer.

Finally, one of the very common applications of cryptographic hash functions is password protection in access control systems (the Unix passwords mechanism is one of the real-life examples). When a user tries to log in to the system, a cryptographic hash of her password is calculated and compared with the one stored in the database. If both hashes are equal, access is granted. This time the essential property in use here is preimage resistance. If someone could read the database and invert hashes stored there, he could bypass the access control mechanism and obtain unauthorised access to the system.

This brief list by no means exhausts possible applications of cryptographic hash functions. Applications we did not mention here include commitment protocols, generic digital signatures, message authentication codes and others. Some further examples with more detailed analysis can be found in section 2.3 of Bart

Preneel's thesis [123]. According to Bruce Schneier [137],

> (..) hash functions are used everywhere. Hash functions are the workhorse of cryptography; they're sprinkled all over security protocols. They're used all the time, in all sorts of weird ways, for all sorts of weird purposes.

Clearly, we can consider cryptographic hash functions as one of the fundamental classes of primitives used in modern cryptography. However, as pointed out by B. Preneel [124], in spite of their popularity, there are not many theoretical results known in this area. Even after eight years this remark remains true and there are much more questions than answers. This makes the study of cryptographic hash functions an exciting research topic with serious practical implications.

Designing cryptographic hash functions There exist three general approaches to designing cryptographic hash functions. The first one is aimed at constructing functions that are provably secure in the sense that the problem of breaking it is related to some well-studied computational problem considered to be very difficult. Since violating specific security property of the function would reveal a way of solving the underlying hard problem efficiently, the security of such functions is considered very well founded. Of course since we do not know whether those problems are hard indeed (in fact we do not even know if $P \neq NP$) this guarantee is still only relative but so far nothing better has been proposed. Classical examples include functions that rely on the hardness of factoring a large composite number proposed by Damgård [44] and Gibson [70] or on difficulty of solving discrete logarithm problem [8]. More recent constructions feature collision-resistant VSH [36] based on a number-theoretic problem related to factoring, one way and collision resistant FSB [4] related to hard problems in coding theory as well as provably one-way MQ-HASH [20] that depends on the difficulty of solving systems of multivariate quadratic equations. Unfortunately, there is also a price to pay for provable security. Most of functions designed that way need longer digests to achieve the desired level of security. They are also relatively less efficient as they usually require complex mathematical operations.

Another approach is to use a trusted block cipher in a special mode that turns the whole construction into a hash function. Functions constructed according to this paradigm also come with a certain type of security proof that relates the security of the hash function to the security of the block cipher. This direction received a lot of attention over the years and the fundamental results in this field are [126, 23]. The idea of reducing the security of the hash to the security of a block cipher has its advantages and drawbacks. The main advantage is that having a block cipher that is considered secure we can construct a hash function that does not have any hidden weaknesses and we can trust it as long as we trust the block cipher. On the other hand, some people feel uncomfortable putting all eggs in one basket and relying on just one primitive (i.e. the block cipher). On the top of that, such constructions suffer from a loss of efficiency compared to the speed of the block cipher [22].

The third approach is to design cryptographic hash functions from scratch. Such designs are called dedicated hash functions. The main advantage of such constructions is their speed and low resource requirements in both software and hardware implementations. Undoubtedly, this made them the most popular class of cryptographic hash functions used nowadays. Examples include such famous functions as MD5 [131] as well as U.S. government standards SHA-1 and SHA-256 [116]. Unfortunately, also here there is a price to pay for this exceptional speed as these designs come without any formal security guarantees. The process of development of such functions can be compared to some evolutionary mechanism. Designers apply "best engineering practises" known so far to create new algorithms and publish them. Published constructions undergo scrutiny by members of cryptographic community who try to develop new attacks. Some of them are quickly broken, other survive longer and may inspire next generations of functions that use best design principles of the existing ones to (hopefully) improve upon them. However, this process cannot be fully relied upon. Even if one could assume that the potential quality of the algorithm can be measured by the man-hours spent on analysis, it is very hard to estimate even how much attention given algorithm received from the academic community. This leaves all the dedicated hash functions at the constant threat of compromise by new clever

analytical techniques.

This thesis The main subject of this thesis is the study of new cryptanalytical techniques applicable to dedicated cryptographic hash functions. Our aim is the development of new cryptanalytical techniques applicable to dedicated heuristic designs. We join the evolutionary environment in which dedicated hash functions are born and eliminated by cryptanalysts trying to develop new attacks on such functions. This approach is not as destructive as one could think at first. Breaking a design or showing some particular weakness in an algorithm enriches our knowledge and indirectly enhances further dedicated designs. Also, we hope that our contribution to analysis of dedicated hash functions will encourage researchers and particularly industry to consider more seriously the first class of designs, i.e. provably secure constructions. We believe that while for some low-risk applications dedicated hash functions are the correct solution, provably secure cryptographic hash functions are the future of hashing for critical applications.

We start our thesis with detailed analysis of security requirements and definitions in Chapter 2. This discussion is quite general and not only introduces various security properties both explicitly and implicitly present in the vast literature on the subject more formally but also tries to organise them and present in a coherent way. We also give some background information on various types of attacks applicable to hash functions.

In Chapter 3 we move on to the first large family of dedicated hash functions, MD5-based designs. We present a unified view on all functions of that class. After describing MD5 and briefly explaining principles of the famous analysis of MD5 performed by the team of Professor Xiaoyun Wang we move to the analysis of SHA-1. We present our contribution to the analysis of this algorithm that allows for finding good differential paths in SHA-1. We conclude the chapter with a section that covers the latest developments in the analysis of SHA-1.

Considering the successful attack on SHA-1, quite a natural question arises about the security of SHA-256 and relatives – another family of NIST-approved hash functions, so called SHA-2 designs. We contribute to the security analysis of those designs developing attacks on two simplified variants of SHA-256. The

results of this research are presented in Chapter 4. Firstly, we show that mixing structures used in SHA-256 are essential to the security of the design as it is possible to launch an attack on the version without them. Secondly, we present our analysis of short variants of SHA-256 with all the modular additions replaced by XORs.

In Chapter 5 we present cryptanalysis of the recently proposed hash function FORK-256 that was meant to be a possible replacement for SHA-256. We present how to construct differential paths in FORK-256 and show how to use them to efficiently obtain near-collisions for this function. We further show that this attack can be extended to find colliding digests faster than by birthday-paradox methods with substantially less computational resources and in a way that is suitable for parallelisation.

In Chapter 6 a more theoretical problem inspired by our FORK-256 analysis is analysed. We investigate how the ability of generating pairs (or tuples) of digests that have a common part influences the chance of finding collisions in different scenarios.

We close the thesis with the consideration of some future research directions.

2

Security requirements and design principles

We dedicate this chapter to the theoretical treatment of fundamental concepts behind hashing. We start with discussing three different approaches to defining computational hardness and discuss their advantages and drawbacks. The notion of computational difficulty is central to defining fundamental cryptographic properties so only after we discuss it we can present fundamental definitions in the appropriate context and discuss their relationships. In practice however, much more is expected from a cryptographic hash function than only preimage and collision resistance and we also review some of those widely assumed properties that are not covered by basic requirements.

In the second part of this chapter we concentrate on attacks on cryptographic hash functions that aim at violating security properties we discussed in the first part. We present a classification of attacks from the point of view of their generality and required resources and we recall most important generic attacks on hash functions. Finally, we present Merkle-Damgård iterative construction, one of the most prevalent design principles used to build cryptographic hash functions,

discuss its fundamental properties and review generic attacks on hash functions designed using this principle. We close this chapter with some alternative approaches that aim at eliminating weaknesses of this method.

2.1 Different notions of computational difficulty

In the previous chapter we informally introduced cryptographic hash functions by saying that they are hash functions satisfying three fundamental security requirements of being preimage resistant, second preimage resitant and collision resistant. In all of these definitions we required that solving the relevant problem should be *computationally infeasible*. However, this is not a precise term and it can mean different things. Depending on the adopted notion of *infeasibility* we get a different model of security. We present here three main directions that formalise the notion of computational hardness and result in three security models for hash functions.

The first notion of infeasibility stemmed from classical complexity theory. In classical analysis of algorithms, one considers how efficiently an algorithm can solve a problem by expressing its running time as the function of the description size of the problem instance. When this complexity function is bounded by some polynomial, the algorithm is considered to be efficient. Note that the complexity function is defined on integers and we usually are interested in asymptotic results, e.g. when problem instances are sufficiently large.

Similar approach to defining collision resistance formally has been used by Damgård [44]. He used infinite families of hash functions, each member h having its own security parameter k and being a function $h : \Sigma^* \to A_k$ where A_k is some finite set. He used the security parameter k as the value corresponding to the instance size in classical complexity theory and proposed the following definition of computationally infeasible problem.

Definition 2.1 (Damgård [44]) *Let $\{C_k\}$ be a Boolean circuit family of polynomially bounded size. Let ϵ_k be the fraction of the instances of size k which are solved by C_k. A problem is* computationally infeasible *if ϵ_k as a function of k vanishes faster than any polynomial fraction.*

Thanks to this notion one can formally define collision resistance in this setting as follows.

DEFINITION 2.2 (DAMGÅRD [44]) *A* family of collision resistant hash functions *is a set of hash functions with the following properties:*

- *There is a probabilistic polynomial time algorithm, which on input value of security parameter selects uniformly and randomly a member of that family with the given value attached.*
- *All functions in the family are computable in polynomial time.*
- *The problem of finding $x \neq y$ such that $h(x) = h(y)$ for a given h in the family is computationally infeasible to solve.*

Above definitions are given here only as an example to illustrate this particular approach. A detailed treatment of this approach is given in [123, Chapter 4].

The fundamental drawback of this notion of hardness is that it requires infinite families of hash functions and obtained results are in a sense asymptotic, while in real life we may require to guarantee the security of some concrete, finite construction.

The need to deal with security assertions without resorting to asymptotics gave raise to a new approach to security called "concrete security". In that framework, hash functions are described as finite families of functions, $H : \mathcal{K} \times D \rightarrow R$, where $\mathcal{K}$ is the set of keys that index the family. The central role in this framework plays the notion of a computational experiment in which an adversary takes part. Basically, we define a sequence of operations requiring some interaction with the adversary and winning conditions for him. Then we define the advantage of the adversary as the probability that he is doing well, i.e. the probability that he wins the game taken over all possible keys and random choices made in the experiment. The algorithm is secure when no adversary with appropriately bounded time and resources can gain significant advantage in the experiment that aims at violating specific security property. To illustrate it on an example, we recall here the concrete security definition of collision resistance from [133].

DEFINITION 2.3 (ROGAWAY, SHRIMPTON) *Let $H: \mathcal{K} \times \mathcal{M} \to \{0,1\}^n$ be a hash-function family and let A be an adversary. Then we define*

$$\boldsymbol{Adv}_H^{Coll}(A) = Pr[K \xleftarrow{\$} \mathcal{K}; (M, M') \xleftarrow{\$} A(K) \ : \ (M \neq M') \wedge H_K(M) = H_K(M')].$$

In the above $M \xleftarrow{\$} \mathcal{S}$ denotes choosing a random element from the distribution $\mathcal{S}$ and calling it M. The authors do not define the threshold for which the advantage of the adversary becomes unacceptable. It depends on the applications, but for collision resistance and functions returning n bit hash one would usually consider $2^{-n/2}$ as the threshold based on the bound derived from the birthday paradox attack.

The concrete security approach is rigorous enough to be used to prove different kinds of security reductions but is still "down-to-earth" so that it is well suited to model real situations. The only issue in the context of hash functions is the necessity of representing them as families of functions. This contrasts with current practice where most of cryptographic hash functions are just single fixed instances rather than families. One could argue though that the faults lies on the side of current practises.

This brings us to the third notion of computational difficulty aimed at formalising hash function properties, recently considered by P. Rogaway in [132] and called "human ignorance model". Instead of requiring that there is no efficient adversary that finds collisions (which is impossible for single-instance functions) we insist that there is no efficient adversary *known to man* that finds collisions for the given function. Since the rest of the construction remains the same, one can reason about different security reductions in a similar manner as in concrete security setting. The only difference is that every result is now related to current human inability of solving a specific problem. This appropriately reflects the current situation in the world of hash functions, but the question of practical value of such assumptions remains open.

2.2 Fundamental properties and their relationships

After this note on different meanings of computational difficulty we can proceed to state more formally definitions of the three fundamental properties of hash functions. Since in this thesis we are mainly concerned with the security of fixed, single instance designs, we recall after [110] definitions in their classical form, essentially compatible with "human ignorance" model.

DEFINITION 2.4 (MENEZES ET AL. [110]) *A hash function h is preimage resistant if for essentially all pre-specified outputs, it is computationally infeasible to find any input which hashes to that output, i.e., to find any preimage x_0 such that $h(x_0) = y$ when given any y for which a corresponding input is not known.*

DEFINITION 2.5 (MENEZES ET AL. [110]) *A hash function h is second preimage resistant if for essentially all pre-specified outputs, it is computationally infeasible to find any second input which has the same output as any specified input, i.e., given x, to find a 2nd-preimage $x' \neq x$ such that $h(x') = h(x)$.*

DEFINITION 2.6 (MENEZES ET AL. [110]) *A hash function h is collision resistant if it is computationally infeasible to find any two distinct inputs x_1, x_2 which hash to the same output, i.e., such that $h(x_1) = h(x_2)$.*

We do not attempt here to present the evolution these notions underwent over the years, for more historical background and discussion of hashing terminology we refer the reader to the recent paper [38].

A natural question that arises is about relations between those notions. We have the following simple observations that establish some of them. The relation between collision resistance and second preimage resistance is straightforward.

FACT 2.1 (MENEZES ET AL. [110]) *Collision resistance implies second preimage resistance of hash functions.*

More complex relationship exists between collision resistance and preimage resistance as we need an additional assumption about the size of the domain and the range.

FACT 2.2 (STINSON [142]) *Suppose $h : \mathcal{D} \to \mathcal{R}$ is a hash function where $|\mathcal{D}|$ and $|\mathcal{R}|$ are finite and $|\mathcal{D}| \geq 2|\mathcal{R}|$. If there exists a Las Vegas algorithm that returns preimages under h of randomly and uniformly distributed $y \in \mathcal{R}$ with probability 1 than there exists a Las Vegas algorithms that finds collisions for h with probability 1/2.*

The extra assumption is significant since when the function does not compress, it may be collision resistant but not one way (a trivial counterexample is the identity function).

To be able to draw more conclusions on relationships between those properties one needs to use more formal definitions of computational infeasibility. A classical result in this area is due to P. Rogaway and T. Shrimpton [133]. Using the concrete security approach they defined seven types of security properties and analysed conventional implications (when one property unconditionally implies another), provisional implications (when one property implies another under some additional assumptions) and separations (when the two are independent). Results of this meticulous study can be summarised in a diagram presented in Fig. 2.1. The three classical notions of security are Pre for preimage resistance, Sec for second preimage resistance and Col for collision resistance. There are also a- and e- variants of some of the notions depending on how the key is chosen (i.e. the instance of the function). Provisional implications, marked by dashed lines depend on some extra assumptions about the size of the domain and range, just as we have seen before. They hold when the function compresses the input significantly (e.g. from 256 bits to 128 bits). Clearly, collision resistance always implies second preimage resistance and when the function is compressing the input enough, it also implies preimage resistance. This shows that the crucial property is collision resistance and understandably this property is the main focus of most security assessments of cryptographic hash functions.

2.3 Other desired properties

Even though cryptographic hash functions emerged as constructions providing the three fundamental properties described before, practical applications gave rise to some additional properties that are often implicitly assumed. When RSA

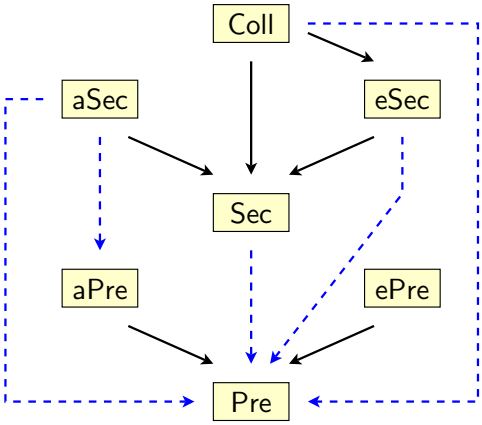


Figure 2.1: Relations between different notions of hash function security [133]. Solid black lines represent classical implications, dashed blue lines mean provisional implications.

public key cryptosystem was invented and first attacks on this scheme were published [47, 54], hash functions were suggested as a tool for preventing such attacks. D. Denning suggested in [52] that the hash function should destroy homomorphic properties of the underlying public key cryptosystem. This notion could be rephrased as the requirement that the hash function should not have any detectable structure or that it "behaves randomly". It can be modeled by picking the hash function as a randomly chosen element of the family of all functions mapping given domain to the given range. This gave raise to two theoretical concepts that try to capture that intuition, random oracles and pseudorandom families.

2.3.1 Simulation of random oracles

A random oracle is a theoretical concept used to describe a perfect random function. A random oracle works by returning a value drawn uniformly and independently from its range for any new input. To make it a function, it returns the same value for inputs that were already queried in the past.

The first mention of such a construction appeared in [63]. Later, Bellare and Rogaway used it to formalise some security notions used in public key cryptography [10] and the random oracle model became an important tool for proving security of cryptographic protocols. Most notably, it was used to prove security of two very important schemes widely used in practice, RSA-OAEP [11] for en-

cryption with RSA and RSA-PSS [12] for RSA signatures. Other results include the proof of security of a parallel signcryption scheme proposed by Pieprzyk and Pointcheval [121]. The main principle of this methodology is to prove that the given protocol is secure when some parts of the protocol are modeled as random oracles. In the real-world implementations, random oracles are replaced by concrete primitives, e.g. hash functions. The whole aim of this approach is to provide some kind of assurance that the protocol does not have security flaws by itself.

However, Canetti et al. showed in [29] that there exist protocols that are provably secure in the random oracle model and inherently insecure when the random oracle is replaced by any concrete hash function. Another, more realistic example was given in [6]. This initiated the trend of avoiding random oracle proofs where possible and stimulated research into new security paradigms that are based on more realistic assumptions (such as intractability of factorisation and solving discrete logarithm, to name the two most commonly used in cryptography). So far, it seems that in spite of some results in this direction [28], there is no universal agreement on what the random oracle emulation could mean.

2.3.2 Hash functions as pseudorandom families

A related attempt at formalising "random behaviour" of hash functions is the use of pseudorandom function families. They were firstly defined in the complexity theoretic setting [71]. The essential idea behind pseudorandom function families (PRFs) is that they are indistinguishable from families of truly random functions (i.e. drawn uniformly from the set of all functions with given domain and range) by adversaries with reasonable resources. This methodology received considerable attention over the years and is an established way of proving a variety of security results [99].

PRF families can be also defined in the concrete security model and this setting was used by Bellare who proved an important result [5] saying that the security of widely used HMAC message authentication algorithm [7] relies solely on the assumption that the compression function of the underlying hash function is a PRF when keyed by the chaining variable input.

Another widespread application in which "random behaviour" of a crypto-

graphic hash function is essential is generation of pseudorandom numbers, as described in NIST standard [66, 67]. The PRNG described there uses SHA-1 and it was shown in [53] that this generator is secure if $f(\ (\cdot + K)\ \bmod 2^n, IV)$ is a PRF family where f is the compression function of SHA-1. The Linux `/dev/random` device also makes use of SHA-1 internally (at least used to in kernel 2.6.10) [72] and Dodis et al. considered in [61] another related application, namely randomness extractors that work properly when underlying compression functions behave like PRF families.

2.3.3 Security of truncated variants

Finally, it is worth to mention one more implicit assumption often made about cryptographic hash functions that hash functions obtained by truncating the output hash to a reduced number of bits are still secure. This means for example that when truncating the n-bit hash functions to only t bits, any collision finding attack on such a reduced variant should require $2^{t/2}$ operations. J. Daemen in his PhD thesis [42] called such functions hermetic. A practical example of the importance of this requirement is the design of SHA-224 [116] that is basically a truncated to 224 bits version of the hash function SHA-256.

2.4 Classification of attacks

An attack on a cryptographic hash function is an algorithm aimed at violating one of the assumed security properties of the function. In the world of cryptographic hash functions a number of distinct notions of attacks exist. In this section we classify them based on their fundamental principles.

The first division is based on the amount of information the cryptanalyst is given and it splits attacks into generic and shortcut attacks.

A generic attack is an algorithm-independent attack that treats the hash function as a "black box" and uses only high-level assumptions about its behaviour like the length of the digest or the distribution of output values.

The opposite of this approach are attacks that depend on the algorithm. A shortcut attack explicitly uses particular structure of the hash function to mount an attack exploiting a design weakness. Most of this thesis, starting from Chapter

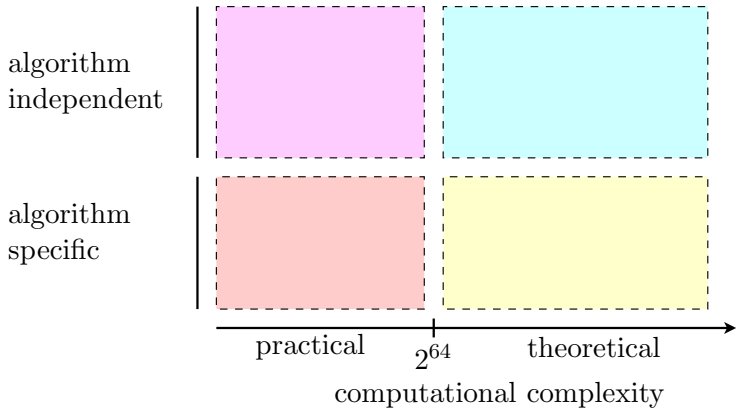


Figure 2.2: Relations between different classes of attacks.

3, is dedicated to shortcut attacks on dedicated hash functions.

Another, independent classification divides attacks into practical and theoretical. This distinction is based purely on the relation of computational complexity of the attack to the current limits of computing technology. Basically, a practical attack is an attack that requires computational resources that are available at the moment. This is opposed to theoretical attacks that need unrealistic resources and thus are impossible to implement with current computing technology. Of course, this means that this distinction is rather fluid and it changes as the technology progresses. The project `distributed.net` showed in 2002 [2] that a complexity of around 2^{64} operations is definitely feasible. Nowadays, this figure is higher, but there is a general consensus that 2^{80} operations is going to be definitely out of reach of technology for considerable time. We can summarise relations between different classes of attacks as a diagram presented in Fig. 2.2.

There is one more remark that should be made while speaking about relations between attacks. Quite naturally, shortcut attacks are interesting only when they are more efficient than the best generic attack available for the particular algorithm and so only those are labelled as "attacks" at all.

2.5 Generic attacks

In this section we concentrate on generic, algorithm independent attacks as they determine upper bounds on complexity of shortcut attacks we focus on in the remaining chapters.

We restrict our attention to fixed functions $h : \mathcal{D} \to \{0,1\}^n$ that produce n–bit long hashes. We introduce the following definition, that will be used later.

DEFINITION 2.7 *We call the function $h : \mathcal{D} \to \{0,1\}^n$ regular, when*

$$\forall y \in \{0,1\}^n \qquad \Pr[\, h(x) = y \,] = \frac{1}{2^n} \ .$$

Note that such a definition is somehow problematic when we want to assume $\mathcal{D} = \{0,1\}^*$ as we are supposed to take the probability over all possible values of x and this value is not well-defined since the domain is infinite. There are two options here. One, more practical, is to restrict the domain to the set of inputs of huge, but fixed length, e.g. $\mathcal{D} = \{0,1\}^{2^{64}}$. This is usually enough as in practice we are also restricted when it comes to input lengths (by padding method and also computational resources). Since a preimage or a collision for a function restricted to inputs of fixed length is also a preimage or a collision for the unrestricted function, this restriction does not impair analysis. More theoretical solution would be to consider a family of functions with different input lengths and use limits at infinity as values for probabilities but this approach seems considerably more complex and also less useful in practice.

2.5.1 Brute-force search

This is the least sophisticated class of attacks and is based on simple evaluation of randomly chosen inputs in the hope of finding a preimage, second preimage or a collision. Assuming that the attacker is able to perform $q = 2^r$ evaluations, the probability that the attacker successfully finds the preimage for a randomly chosen output y is 2^{r-n}. This result should be taken into account when designing appropriate output length of a hash function as for too short digests this trivial attack may be or become feasible. When we pick just one point, this probability is 2^{-n} and we speak about a random attack.

2.5.2 Birthday paradox attack

We can do much better in case of the problem of finding collisions. Instead of generating pairs, comparing hash values and later discarding them if they do

not match, we can store generated results and use them to find collisions more efficiently. More precisely, we use the following procedure

$x_1 \xleftarrow{\$} \mathcal{D}$
$y_1 \leftarrow h(x_1)$
for $i = 2$ to q **do**
  $x_i \xleftarrow{\$} \mathcal{D}$
  $y_i \leftarrow h(x_i)$
  if $y_i = y_j$ and $x_i \neq x_j$ for some $j \in \{1, \ldots, i-1\}$ **then**
    return (x_i, x_j)
  end if
end for
return failure

where $\xleftarrow{\$}$ means drawing an element from the set randomly and uniformly.

It is easy to show that if the function is regular, the probability p_q of success of this algorithm can be estimated as

$$p_q = 1 - \prod_{i=1}^{q-1} \frac{2^n - i}{2^n} = 1 - \prod_{i=1}^{q-1} \left(1 - \frac{i}{2^n}\right) \approx 1 - \prod_{i=1}^{q-1} e^{-i/2^n} \approx 1 - e^{-\frac{q^2}{2^{n+1}}} \ .$$

It follows that when we use $q = \sqrt{2} \cdot 2^{n/2}$ values we have a good chance (approx. 0.63) of finding a collision.

If we modify this procedure and keep running until we find a collision, the expected number of evaluations of the function is $\sqrt{\pi/2} \cdot 2^{n/2}$ (cf. [146, Appendix A]).

A variant of this approach has been considered by G.Yuval [163] who showed how by generating two sets of variants of a message one can exploit hash collisions to obtain identical signatures of two different documents. This and similar scenarios have been studied in more detail in [118].

It seems that to find a collision, one would need to store around $2^{n/2}$ hashes, however using a method proposed by J.-J. Quisquater and J.-P. Delescaille [127, 128] one can achieve the same asymptotic speed but much smaller storage requirements. The method uses the restriction of the hash function h to a domain equal to the range to generate a random walk in the space of digests by iterative

application of the function, i.e. the procedure described as

$$x_0 \xleftarrow{\$} \{0,1\}^n,$$
$$x_i \leftarrow h(x_{i-1}), \quad \text{for } i \geq 1.$$

For sufficiently large domain sizes, the restricted map is almost never bijective (what comes from the fact that the number of bijections is $2^n!$ and the number of all functions is 2^{n2^n}), so we can expect collisions to exist. Now, since the set of values is finite, there exist some value $l \in \mathbb{N}$ such that $h(x_{l+c}) = h(x_l)$ but $x_{l+c} \neq x_l$. This is a collision. The trick is to recognise one when it happens. Quisquater and Delescaille used so called distinguished points method to store only a small fraction of points on the path. When a collision is detected in the stored table of distinguished points and verified as a real collision, it is enough to reconstruct the last part of the path from the last distinguished point. This allows to decrease memory requirements dramatically (in the original paper [127] distinguished points that have 20 MSB bits equal to zero were used, so the gain was of factor 2^{20}). This method can be also efficiently parallelised as shown in [146]. This smart approach made collision search attacks more feasible in practice, however, asymptotically, it still needs an exponential amount of storage memory, but with a very small constant.

So far, for the sake of simplicity, the above analysis assumed that the hash function is regular. However, this may not be the case. An important study into hash functions which are not regular was performed recently by M. Bellare and T. Kohno [9] who investigated what is the impact of "regularity" of the function on complexity of birthday attacks. For a hash function $h : \mathcal{D} \to \mathcal{R}$ where $\mathcal{R} = \{R_1, R_2, \ldots, R_r\}$ they defined the parameter called balance as

$$\mu(h) = \log_r \left[\frac{d^2}{\sum_{i=1}^{r} d_i^2} \right] ,$$

where $d_i = |h^{-1}(R_i)|$ and $d = |\mathcal{D}|$. This parameter is always a real number between 0 and 1 and characterises the regularity of the function, with $\mu(h) = 1$ for a regular function and $\mu(h) = 0$ for the worst case of a constant function. They proved a number of quantitative results showing that finding collisions might be

significantly faster for functions of low balance.

2.6 Iterated hash functions

It is not easy to design a function that can process arbitrary, or at least huge and unknown in advance, amounts of data. The natural approach is to use some kind of iterative process that processes data in chunks. This solution was firstly suggested in R. Merkle's PhD thesis [111]. The main idea is to use a compression function, i.e. a function that maps longer, but fixed size intputs to shorter outputs. In other words a compression function is a function

$$f: \{0,1\}^{m+n} \to \{0,1\}^n$$

where $m > 0$. Then one can use it to hash arbitrary length data using the following process, independently studied by R. Merkle [112] and I.Damgård [45].

INPUT: message $M \in \{0,1\}^*$

OUTPUT: digest $h(M) \in \{0,1\}^n$

- $M := M||1||0^z$ where z is the smallest integer s.t. m divides $|M| + 1 + z$.
- (optionally) Append a block of m bits with encoded initial length of the message M.
- Split the message M into blocks of length m, $M = (M_0, M_1, \ldots, M_b)$.
- Perform the iterative compression

  $H_0 := IV$ {quite often $IV = 0^b$}

  for $i = 0$ **to** b **do**

  $$H_{i+1} := f(H_i||M_i) \tag{2.1}$$

  end for

- Return the result $h(M) = H_{b+1}$.

This process is illustrated in Fig. 2.3. A fundamental property of this construction proved in [112, 45] assures that as long as the padding contains the length of the initial message the whole hash function is collision resistant as long as the compression function is collision resistant. Since this result reduces the problem

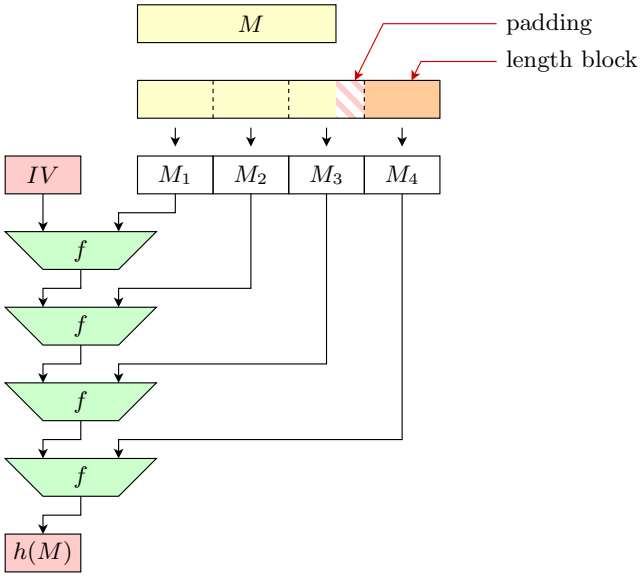


Figure 2.3: Iterative application of a compression function f yields a construction that can be used to hash messages M of arbitrary length.

of designing secure hash functions to the more manageable problem of designing secure compression functions, this design principle has been widely adopted and currently vast majority of published hash functions use the iterative construction.

2.6.1 Weaknesses of iterated hash functions

It turns out that in spite of the security reduction concerning collision resistance, iterative cryptographic hash functions have a number of other, more subtle weaknesses, especially when the reference model is a perfect hash simulated by a random oracle. We briefly mention most important results in this section.

Multicollisions One can think about a generalisation of the notion of a collision from pairs to r–tuples of messages. We get the following, straightforward definition.

Definition 2.8 *An r–multicollision for the cryptographic hash function h, where $r \geq 2$, is an r–tuple $(M_1, M_2, \ldots, M_r)$ of messages such that $M_i \neq M_j$ for $i \neq j$, $i, j \in \{1, \ldots, r\}$ and*

$$h(M_1) = h(M_2) = \ldots = h(M_r) \ .$$

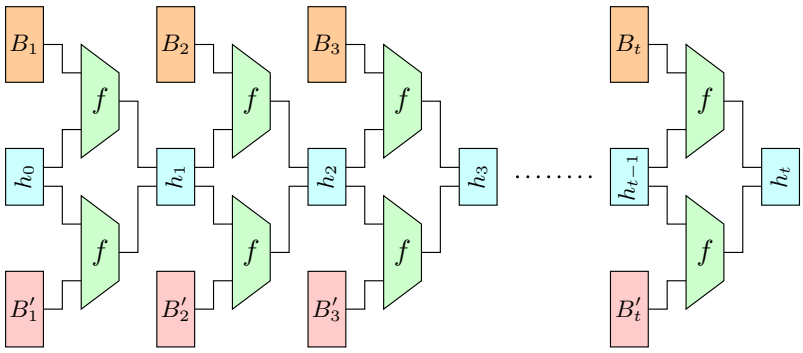


Figure 2.4: For an iterated hash function, a 2^t–multicollision can be constructed from a sequence of t compression function collisions.

When we consider a perfectly random hash function, finding an r–multicollision requires around $2^{n(r-1)/r}$ evaluations of the hash function. When r is large, this value is approaching 2^n and thus it is significantly harder to find multicollisions than ordinary collisions that require around $2^{n/2}$ of work. However, in 2004 A. Joux showed [84] that finding multicollisions is significantly easier for iterated hash functions. He observed that after finding t pairs of message blocks $(B_1, B'_1), \ldots, (B_t, B'_t)$ that yield collisions for the compression function f one can easily generate 2^t t-block messages b that hash to the same digest by setting $b = (b_1, b_2, \ldots, b_t)$ where b_i is either B_i or B'_i. This idea is illustrated in Fig. 2.4. The total complexity of finding such 2^t–multicollisions is of order $t2^{n/2}$, significantly less than for an ideal hash function.

Second preimages faster than 2^n In an ideal hash function with n bit output modeled as a random oracle, finding a second preimage requires around 2^n calls to the hash function. We would like any hash function to have this behaviour, but R. Dean observed that this property is not satisfied for iterated hash functions as long as it is easy to find fixed points in the compression function [51]. The problem of finding second preimages has been also studied by J. Kelsey and B. Schneier who showed in [88] that it is possible to do it faster than 2^n operations even without the need for finding fixed points. Their main result shows that for long messages, consisting of 2^k blocks it is possible to find second preimages with work factor $k \cdot 2^{n/2} + 1 + 2^{n-k+1}$.

The idea is based on so called expandable messages which are sets of pairs of

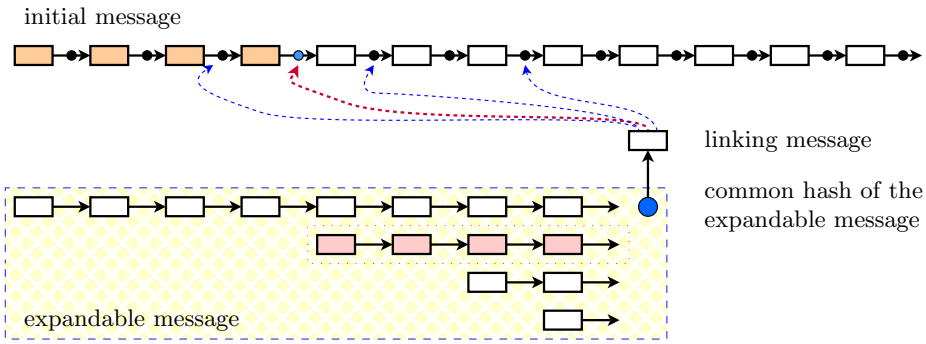


Figure 2.5: After generating an expandable message one can find a second preimage for a long message by trying to link the output hash of the expandable message with one of the chaining values of the initial long message.

messages of differing lengths that can be used to construct messages of varying lengths and at the same time producing the same hash value before the application of the final block that contains the length of the message. They can be seen as generalisation of multicollision, where instead of just a single message block one can use more blocks as long as the hash state collides at the end.

Having such an expandable message that can produce messages of lengths from k blocks to $2^k + k + 1$ blocks one can find a second preimage for a long message $m = (m_0, m_1, \ldots, m_{2^k+k})$ consisting of $2^k + k + 1$ blocks. Basically, we store all the intermediate chaining values and then try to link the final hash of the expandable message with one of the chaining values of the original message. When we find a match we can always adjust the expandable message to make it the same length as the first part of the initial message from the start to the point where the match was found and thus obtain two long messages of the same length that collide at some point. Then the expandable message is appended with the rest of the initial message to yield the same final hash since hashing the length block will give the same result now. This procedure is illustrated in Fig. 2.5. The total complexity is equal to $k \cdot 2^{n/2+1} + 2^{n-k+1}$. The first term is due to the generation of the expandable message, the second one describes the complexity of finding the link between expandable message and one of the chaining values. In practice, this means that for SHA-1 and a very long message of about 2^{60} bytes one can find second preimages in 2^{106} effort rather than the expected 2^{160}.

Distinguishing iterative hash functions Another specific feature of iterated hash functions is the possibility to distinguish them from random oracles even if the compression function f is a random oracle itself. If the scheme does not include the block length and we have access to the compression function, we can easily do this in the following way, described in [40]. We first query the hash function h to get $u = h(m_1)$ where m_1 is a single block message. Then we query the compression function to get $v = f(u, m_2)$. In the final step we query the hash function again to get $z = h(m_1||m_2)$ and compare the output z with v. If they are identical we are almost sure that the hash function h is an iterated hash function based on the compression function f.

Even if we have access to only the whole hash function h we still can tell it apart from a random function using the birthday attack and the message extension property. In the first step, we look for two different one-block messages m_1, m_1' that collide under the hash function h. Then, we pick another random block m_2 and test whether $h(m_1||m_2) = h(m_1'||m_2)$. For an iterated hash functions those values will be identical, for a truly random function h the probability of this equality is negligible. In fact, this message extension property has been used in some practical attacks that are feasible once we can find even a single collision.

Iterated construction does not preserve balance As we have already mentioned, [9] deals with analysis of the impact of function's balance on its generic properties. The paper also considers an interesting problem of balance preservation that asks whether the iterated hash function based on a regular compression function is also regular. This can be seen as an analogue to the preservation of collision resistance proved by Merkle and Damgård. Unfortunately, the answer is no, as shown by the following counterexample. Let the compression function $f : \{0,1\}^{n+m} \to \{0,1\}^n$ be defined as $f(c||m) = c$. It is regular since each point in the range $\{0,1\}^n$ has a preimage of size 2^m. However, when this function is used to build an iterated hash function h, for all input messages the value will be IV, the initial value used to start the iteration and we end up with a constant function that has the worst balance equal to zero.

2.6.2 Alternative constructions

Due to weaknesses described above, researchers are looking for alternative ways of constructing hash functions out of smaller building blocks that do not have weaknesses of the Merkle-Damgård design.

Wide pipe As a possible means of strengthening the classical Merkle-Damgård iteration against some of the attacks, S. Lucks proposed a construction called wide pipe [100]. He considered hash functions built from two compression functions, $C' : \{0,1\}^{w+m} \to \{0,1\}^w$ and $C'' : \{0,1\}^w \to \{0,1\}^n$ where $w > n$ is a parameter denoting the size of the hash function state. The modified iteration uses the compression function C' with wider state and only after hashing the last message block, the final transformation C'' is applied that reduces the size of the state to n bits, as presented in Fig. 2.6. Assuming that the final iteration is secure (i.e. the composition $C'' \circ C'$ is collision resistant), extended state prevents some of the attacks such as multicollisions or second-preimages. The drawback of this approach is that it uses a larger state that may result in decrease of efficiency and it looks like the weakest part cryptanalysts could concentrate on would be the final compression function C''.

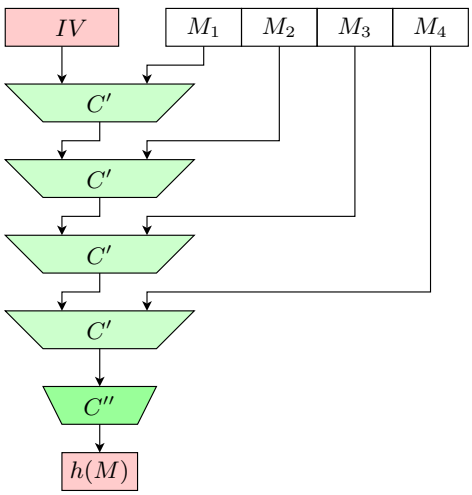


Figure 2.6: Wide pipe construction prevents some of the attacks on iterated hash functions by using a larger intermediate state of w bits handled by compression function C' that finally gets reduced to n bits by another compression function C''.

Randomised Hashing An interesting improvement aiming at strengthening the MD construction was recently presented by Halevi and Krawczyk [73]. Their construction uses the classical Merkle-Damgård iteration without any structural changes but introduces two ways of randomising the message. In the first one, each message block M_i is XORed with a random block r (sometimes called the "salt") before feeding it to the compression function. In other words, the iteration (2.1) becomes

$$H_{i+i} := h(H_i \mid\mid M_i \oplus r) \ .$$

The other scheme also uses a random block r but it also prepends it to the message while still performing XOR with r for all message blocks. This is equivalent to using the first scheme on the modified message equal to $\mathbf{0}||M$ where $\mathbf{0}$ denotes a zero block. Such constructions can be proved to be target collision resistant (TCR) and the second one satisfies even stronger notion of enhanced TCR, defined in [73], under relatively weak assumptions that the compression function satisfies a specific requirement very closely tied to the second preimage resistance. Thanks to that, digital signature schemes that rely on the hash function being TCR can be used safely with this modified MD construction even if the underlying compression function is not collision resistant but satisfies only a weaker security property related to second preimage resistance.

HAIFA Another approach was proposed by Biham and Dunkelman [16]. They suggest that the input to the compression function f should be augmented by two more parameters: the number of bits hashed so far and "salt". Thus, the compression function in HAIFA is defined as $f : \{0,1\}^{n+m+b+s} \rightarrow \{0,1\}^n$ and the iteration is modified to take the following form

$$H_{i+1} := f(H_i \mid\mid M_i \mid\mid \#bits \mid\mid salt) \ .$$

This construction foils multicollision and second preimage attacks but poses a new challenge to designers of compression functions, namely how to integrate the additional information in a secure manner. There is always a risk that adding new parameters may introduce some new weaknesses we are not aware of at the moment. It is interesting to note that according to the recent work [17],

HAIFA framework can accommodate for randomised hashing as well as some other recently proposed modes of operation.

3C constructions Finally, Gauravaram et al. [68] proposed a class of designs with a more specialised security goal, aimed at strengthening the Merkle-Damgård construction against multi-block collision attacks similar to Wang's. They propose to use an additional state variable called accumulator chain that is updated after each step of the compression function with data derived from the main hash function state. Finally, at the end of the iterative process this accumulated value is fed into the last compression function (that may be different from the one used in the main iteration). Such a construction has been adopted in a recently designed hash function Maelstrom-0 [64].

2.7 Summary

In this chapter we outlined most important security requirements and design principles that are relevant to the analysis of cryptographic hash functions. The literature on the theory of hashing is vast and we did not attempt to present a systematic treatment of this subject. Our aim was only to highlight theoretical results that influence current design practices and establish basic context in which our cryptanalytical results contained in the following chapters can be better understood.

3

Analysis of hash functions of the MD/SHA-1 family

The vast majority of dedicated hash functions published up to date is more or less designed using ideas inspired by functions MD4 and MD5. Many functions like HAVAL [164], RIPEMD [27], RIPEMD-160 [125] but also SHA-0 [65] and SHA-1 [115] and similar designs like HAS-160 [145] or HAS-V [119] all exhibit strong resemblance. We start this chapter with presenting all those different functions within a single framework and stress their common properties. Then, we present a brief survey of the history of cryptanalytical results related to MD4 and MD5, starting from the oldest ones to the most recent improvements of seminal work of X. Wang. After that we move to the two designs proposed as U.S. Information Processing Standards, SHA-0 and SHA-1. After introducing the structure of both functions we recall the first differential attack on the former function and present our investigation into the possibility of extending this approach to SHA-1. We present a novel characterisation of the message expansion algorithm as a linear

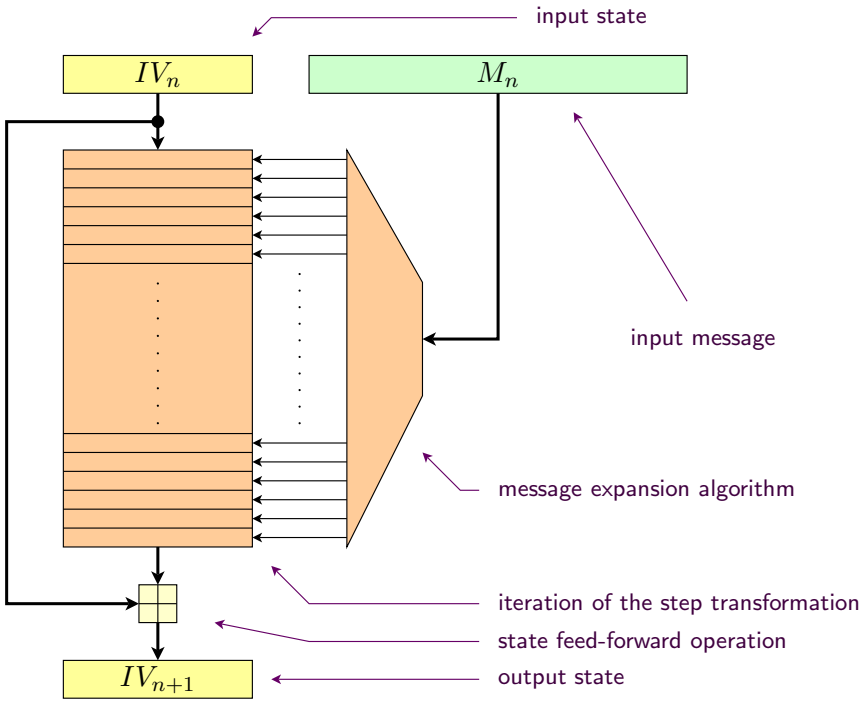


Figure 3.1: Functions of the MD/SHA family have compression functions based on the construction consisting of a message expansion algorithm, iteration of the step transformation and state feed-forward operation.

code over $\mathbb{F}_2$ and study its properties. Thanks to coding theory tools we obtain some upper bounds on weights of differential characteristics in SHA-1 and prove lower bounds useful for short variants of SHA-1. We finish this chapter with a review of recent rapid developments in the analysis of this function and some proposals to counter newest attacks related to our approach.

3.1 Architecture of the MD/SHA-1 family

When comparing compression functions of aforementioned hash functions it is easy to observe that all of them have the basic structure presented in Fig. 3.1. The three fundamental parts are the iteration of the step transformation, the message expansion algorithm and the state feed-forward operation. There is no information available as to why MD4/MD5 was designed that way, which in turn influenced many later designs, but one can identify some general principles behind such design choices.

General structure. The basic observation is that MD/SHA compression functions can be seen as a generalization of the Davies-Meyer hashing mode [126] in which a block cipher is replaced by sequence of transformation steps and the XOR addition is replaced by the modular addition in the feed forward.

The classical construction with XORs yields a secure collision-resistant function as long as the underlying block cipher is secure [23] so it seems to be a reasonable decision to design the hash function using this mode. Of course the security proof is only a reduction, to get a secure hash function one needs a secure block cipher.

All underlying block ciphers that are used in MD-like constructions are based on the iteration of a bijective transformation of the input state that is parametrized by a single word of the expanded message. The general structure of step transformation is presented in Fig. 3.2. Such a construction can be seen as an extension of the well-known classical Feistel structure and according to [138] it can be classified as a source-heavy Unbalanced Feistel Network.

The main advantage of such an extension is the possibility of tailoring the structure of the cipher to the most common computing platform, i.e. personal computers equipped with 32-bit processors. When a single block size is set to 32-bits, the whole extended Feistel network can be realised very efficiently because it mainly deals with whole 32-bit words of data and such instructions are particularly efficient. When the function f is a bit-wise Boolean function (as is the case in all MD-like designs), we obtain a particularly efficient structure.

Since in source-heavy Feistel networks propagation properties are rather weak as just a single word of the state is affected after one step, to obtain thorough mixing a large number of such steps is required. This is why underlying ciphers of functions like MD5 or SHA use 64 or 80 rounds, much more than in conventional ciphers that employ rounds with much better avalanche properties. However, this larger number of rounds is still outweighed by an exeptional speed of the step transformation and in the end, those designs are very fast.

Message expansion algorithms. The message expansion process (that could be called "key-scheduling" in the context of block ciphers) is very simple in all the MD-like constructions. Most of the functions use just a collection of carefully

selected permutations of message words. For example, MD5 in its 64 steps uses identity permutation $\sigma_1 = id$ for the first 16 steps $0, \ldots, 15$, $\sigma_2 : k \mapsto (5 \cdot k + 1) \bmod 16$ as the second one used in steps $16, \ldots, 31$, $\sigma_3 : k \mapsto (3 \cdot i + 5) \bmod 16$ as the third one and $\sigma_4 : k \mapsto 7 \cdot i \bmod 16$ as the last one used in steps $48, \ldots, 63$. The importance of selecting the right permutation may be illustrated on the example of RIPEMD. While the first version was shown to have some weaknesses [58], the improved design, where much attention has been paid to the selection of message permutations [125], has not been successfully attacked.

A little more sophisticated approach was presented by the designers of SHA-0 who employed a linear recurrence relation (that could be seen also as a word-based linear feedback shift register) that produces a new word of the expanded message based on the last 16 words produced previously. SHA-1 introduced another variation of this theme with slight modification that rotates the newly obtained word by one bit. This of course is costlier than the simple permutation of words and some other designers tried to settle for the middle ground: HAS-160 [145] and HAS-V [119] mix 16 permuted input words with 4 extra words being linear combinations of some of the input words to obtained 20 words used in 20 consecutive steps of the compression function.

Attacks on underlying block ciphers When the compression function is stripped of its Davies-Meyer mode of operation, what is left is essentially a classical block cipher.

Such a cipher was proposed explicitly by H. Handschuh and D. Naccache and called SHACAL. It is based on SHA-1 and submitted to the NESSIE competition [74]. The design received a detailed security analysis [75]. This created a considerable interest in analysis of such block ciphers. The first results came two years later when E. Biham et al. presented the rectangle attack on 49-round reduced version of SHACAL-1 [18] and M. Saarinen considered related-key attacks on block ciphers derived from functions MD5 and SHA-1 [134]. Over the years SHACAL-1 received considerable attention with studies focusing on related key attacks [90, 98, 62, 19]. Interestingly, there are no indications that results of analysis of underlying block ciphers facilitated analysis of collision-resistance of SHA-1. On the contrary, the first result for the full SHACAL-1 [62] describes the

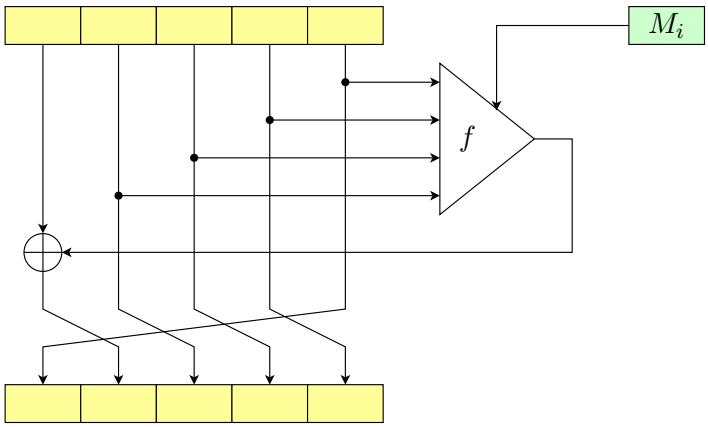


Figure 3.2: Step transformations of all MD-like functions are based on source-heavy Unbalanced Feistel Networks

related-key scenario and uses results of cryptanalysis of the hash function SHA-1 discovered by Wang et al..

Other results showed non-randomness of the cipher based on HAVAL [158] and on MD4 and MD5 [89].

3.2 History of attacks on MD4, MD5 and relatives

The first member of the MD family was MD4 [130]. Just a year after its publication in 1990, an attack on the last two out of three rounds (i.e. last 32 out of 48 steps) has been presented [24]. This motivated R. Rivest to present the improved version, called MD5 [131]. Later, Vaudenay showed that the first two rounds of MD4 are not collision-resistant and it is possible to get near-collisions for the full MD4 [150].

The first attack on MD5 came in 1993 [25]. Den Boer and Bosselaers showed that it is possible to find pseudo-collisons for the compression function of MD5, i.e. they showed a way of finding two different values of the IV and a common message M such that $MD5compress(IV, M) = MD5compress(IV', M)$. This did not threaten the usual applications of MD5, since in normal situations one cannot control inputs of chaining variables. However, a recent result by Contini and Yin showed that such differential paths have applications in attacks on HMAC and NMAC constructions where the compression function keyed by the IV values should be a PRF family [39].

A major step forward in the analysis of MD-based designs was made by H. Dobbertin who around 1996 developed a general method of attacking designs similar to MD4. His method aims at finding collisions and is based on describing the function as a system of complicated, non-linear equations that represent the function. The core of this approach is a method of deriving specific constraints on the values of variables that make the big and diffucult system of equations possible to solve. With his method he succesfully attacked MD4 showing that one can find collisions using computational effort of around 2^{20} hash evaluations [55, 59] and was able to show that the two first rounds of MD4 are not one-way [60]. He also exhibited weaknesses in RIPEMD [58] and showed collisions for the compression function of MD5 with a chosen IV [56, 57]. An extensive treatment of his method, particularly in the context of MD5 analysis can be found in the PhD thesis of M. Daum [46].

Dobbertin's method was also applied to the analysis of HAVAL [164]. Some initial results were obtained by Kasselman and Penzhorn [87] who showed collisions for the last two rounds of three-pass HAVAL, Park et al. [120] who analysed first two and last two rounds and by Her and Sakurai with their analysis of the first and third round [79]. Finally, B. van Rompay et al. presented cryptanalysis of the full 3-pass version [148, 147].

A completely new chapter in the history of analysis of MD-like designs was opened in August 2004 when X. Wang et al. announced a novel, more efficient and flexible method of finding collisions in such designs [152]. Her rump session presentation during CRYPTO'04 in which she claimed it is possible to find collisions in MD4 by hand [153] was welcomed with a round of applause. The details were subsequently presented in two papers, one focusing on MD5 [155] and the other dealing with MD4 and RIPEMD [151]. Wang's method is essentially a differential attack in which both kinds of differences, XOR and modular, are taken into account and considered simultaneously. This is essentially equivalent to keeping track of signed binary differences (when one not only considers a change in the bit, but also the direction of the change, either $0 \rightarrow 1$ or $1 \rightarrow 0$). This gives more information on the state of the difference and allows for more control over the propagation of differences through the function. Another important element

was a method of finding messages satisfying so-called sufficient conditions that ensure correct propagation of differences. The non-obvious part of the attack is finding the appropriate differential path and some independent research has been dedicated to that problem [136, 46]. Since 2004, Wang's method has been studied intensively (with [77] being one of the most extensive treatments) and many enhancements have been proposed to both attack on MD4 [114, 136] and MD5 [91, 92, 93, 140, 21]. Latest results are so efficient that computing a pair of colliding messages for MD4 is no more difficult that evaluating the function itself [135]. It has been also shown how to find second preimages for MD4 with computational effort of around 2^{27} MD4 [160]. Wang's method was also used to attack longer versions of HAVAL with 4 and 5 passes [161].

Since Wang's attacks allow for finding collisions not only for the compression function of MD5 but for the whole hash function with the predefined IV (in fact, it is possible to find collisions for any given IV), such collisions have significant real-life impact. Lucks and Daum showed that it is possible to craft two postscript files that display completely different text while having the same MD5 digest [101] and Gebhardt et al. generalised this method to other formats [69]. Lenstra and de Weger showed how to construct two different X.509 certificates that contain identical signatures [96] and recently, they improved that result with Stevens to two X.509 certificates with different `Distinguished Name` fields that have identical signatures [141].

3.3 Finding good differential patterns for SHA-1

The first version of the Secure Hash Algorithm (SHA) was presented by NIST in 1993 [65]. Two years later, the function was slightly modified and an updated version of the standard was issued [115]. Although no details were made public, the claim was that the improvment removed a technical weakness in the algorithm, most likely discovered by cryptanalysts from NSA.

Indeed, in 1998 F. Chabaud and A. Joux presented a differential attack on the initially proposed function, SHA-0, that can be used to find collisions with complexity of 2^{61} hash evaluations [33]. This attack was later implemented and refined allowing for finding an actual collision [83, 95]. Since SHA-0 and SHA-1

are different by a small (but significant) change in the message expansion algorithms, it is quite natural question to ask whether it is possible to extend the original attack of Joux and Chabaud to the improved design of SHA-1. Due to the same round structure, the same technique used to attack SHA-0 could be applied to launch an attack on SHA-1 provided that there exists a good enough differential pattern. In this section we present results of our investigation into the existence of such patterns in SHA-1. Most of the contents is based on our paper [104].

3.3.1 Description of compression functions of SHA-0 and SHA-1

The compression functions of both SHA-0 [65] and SHA-1 [116] hash 512-bit input messages to 160-bit digests. First, 512 bits of the message are divided into 16 32-bit words W_0, $W_1, \ldots, W_{15}$. The rest of 80 words is generated out of the first 16 words using message expansion algorithms which constitute the only difference between both designs. In SHA-0, expanded words are generated according to the following recursive formula

$$W_i = W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16} \quad \text{for} \quad 16 \leq i \leq 79 \ , \tag{3.1}$$

while SHA-1 uses the formula

$$W_i = ROL^1(W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \quad \text{for} \quad 16 \leq i \leq 79 \ , \tag{3.2}$$

where $ROL^k(X)$ denotes a rotation of the word X by k positions left. This process is illustrated in Fig. 3.3.

If this is the first application of the compression function, five 32-bit registers A, B, C, D, E are initialized to values $A_0 =$ `0x67452301`, $B_0 =$ `0xefcdab89`, $C_0 =$ `0x98badcfe`, $D_0 =$ `0x10325476`, $E_0 =$ `0xc3d2e1f0` accordingly.

Next, the algorithm applies 80 steps ($i = 0, \ldots, 79$). Each step is of the

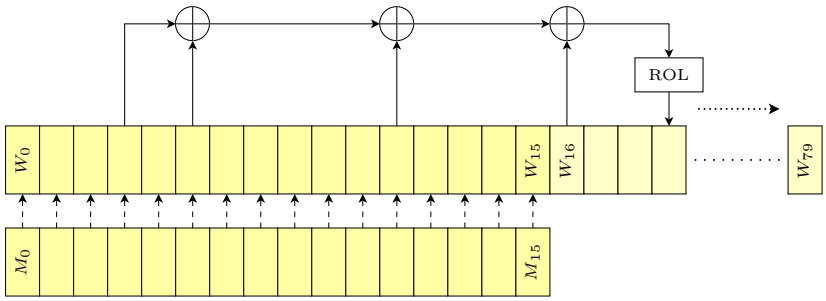


Figure 3.3: Message expansion algorithm of SHA-1 produces 80 words of the expanded message out of 16 initial message words. SHA-0 uses the same structure but without the final rotation left (ROL) of the new word.

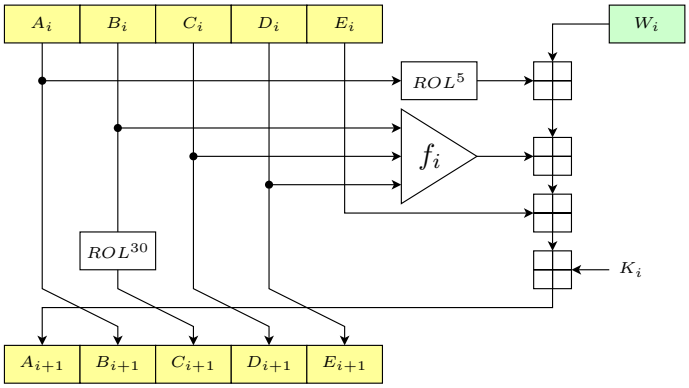


Figure 3.4: The step transformation of functions SHA-0 and SHA-1.

following form:

$$
\begin{aligned}
A_{i+1} &= ROL^5(A_i) \boxplus f_i(B_i, C_i, D_i) \boxplus E_i \boxplus W_i \boxplus K_i \ , \qquad (3.3)\\
B_{i+1} &= A_i \ , \\
C_{i+1} &= ROL^{30}(B_i) \ , \\
D_{i+1} &= C_i \ , \\
E_{i+1} &= D_i \ ,
\end{aligned}
$$

where $\boxplus$ denotes addition modulo 2^{32} and A_i, B_i, C_i, D_i and E_i denote the values of the registers after i-th iteration. This transformation is presented in Fig. 3.4. Functions f_i and constants K_i used in each iteration are given in Table 3.1.

Finally, the output of the compression function is the concatenation of bits of $A_0 \boxplus A_{80}$, $B_0 \boxplus B_{80}$, $C_0 \boxplus C_{80}$, $D_0 \boxplus D_{80}$ and $E_0 \boxplus E_{80}$.

Table 3.1: Functions and constants used in SHA-0 and SHA-1.

step number i	$f_i(B,C,D)$	K_i
0 – 19	$(B \wedge C) \oplus (\neg B \wedge D)$	0x5a827999
20 – 39	$B \oplus C \oplus D$	0x6ed9eba1
40 – 59	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$	0x8f1bbcdc
60 – 79	$B \oplus C \oplus D$	0xca62c1d6

3.3.2 Differential Attack of Chabaud and Joux

Chabaud and Joux presented in [33] a differential attack on SHA-0. The fundamental observation they made is that a change in the j–th bit of the word W_i can be corrected by complementary changes in the following bits:

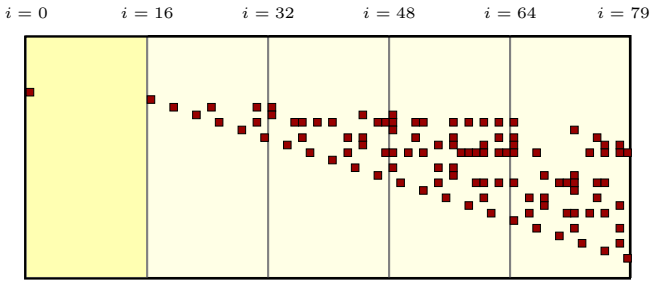
- bit $(j+6) \bmod 32$ of W_{i+1},
- bit j of word W_{i+2},
- bit $(j+30) \bmod 32$ of W_{i+3},
- bit $(j+30) \bmod 32$ of W_{i+4},
- bit $(j+30) \bmod 32$ of W_{i+5},

provided that functions $f_{i+1}, \ldots, f_{i+4}$ and modular additions $\boxplus$ behave like linear functions, that is, a single change of the input to f results in a change of the output of f, a change in two inputs of f leaves the result unchanged and differences propagate through additions without carries. They showed that a one bit disturbance can be corrected by such a pattern with probability between 2^{-2} and 2^{-5} depending on functions $f_i, \ldots, f_{i+4}$, if the disturbance is introduced in the second bit ($j = 1$).

If a disturbance is introduced in the position $j \neq 1$, then there is an additional factor of 2^{-3} caused by 50% chance of inducing a carry in additions in steps $i+3$, $i+4$, $i+5$.

The attack is possible due to the property of the message expansion function which does not mix bits in different positions. Thanks to that it was possible to consider the message expansion algorithm as a bit-wise one. Enumeration of all 2^{16} possible bit patterns in the position 1 allowed for choosing a disturbance

Figure 3.5: Propagation of one bit difference in SHA-1 message expansion.



pattern in the first bit position that led to a global differential pattern δ producing a collision with probability 2^{-61}.

Remark. It is possible to improve the attack of Joux and Chabaud by reducing probabilistic behaviour of some initial corrections using a better strategy of selecting messages rather than picking random ones. Biham and Chen proposed in [13] the method of so-called neutral bits. They showed that having a message that behaves correctly for at least 16 first steps after adding a difference δ, it is possible to construct a big set of pairs $(M, M \oplus \delta)$ that have much better probability of a successful correction than the pairs produced from random messages.

3.3.3 Analysis of the message expansion algorithm of SHA-1

An additional rotation in the message expansion formula (3.2) makes finding corrective patterns used in [33] impossible, because now differences propagate to other positions. For SHA-1, a one-bit difference in one of the 16 initial blocks propagates itself to at least 107 bits of the expanded message W. This is illustrated in Fig. 3.5. However, we were able to find a difference pattern with only 44 bit changes in the expanded message. This suggests that it is interesting to investigate the message expansion algorithm of SHA-1 in a greater detail and check to what extent the differential attack can be applied also to SHA-1.

The important property of the message expansion process given by the formula (3.2) is that it is a bijective function producing 16 new words out of 16 old ones. This implies that it is possible to reconstruct the whole expanded message given any 16 consecutive words of it, in particular the first 16. Moreover, if we

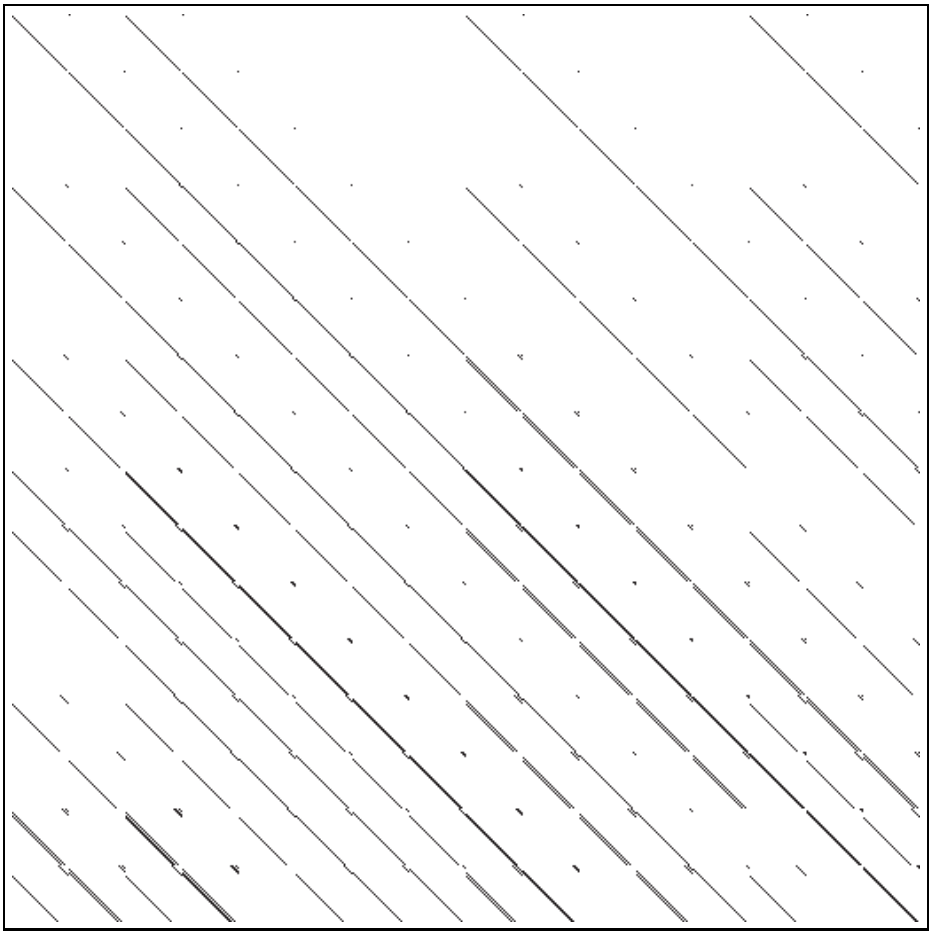


Figure 3.6: Illustration of the matrix of the linear transformation A induced by the message expansion process of SHA-1. The matrix has dimensions 512×512 and each tiny box corresponds to a bit equal to 1 while the rest are zeros.

consider it on a bit level as a function $A : \mathbb{F}^{512} \to \mathbb{F}^{512}$, it is easy to see that A is $\mathbb{F}_2$-linear as the only operations used are word rotations (which are permutations of bits) and bitwise XOR operations. Transformation A can be represented as a binary matrix of dimensions 512×512, depicted in Fig. 3.6. The expansion of the initial message $m \in \mathbb{F}^{512}$ (we consider m to be a column vector) can be expressed as a long vector

$$E_1(m) = \begin{bmatrix} m \\ \hline A(m) \\ \hline A^2(m) \\ \hline A^3(m) \\ \hline A^4(m) \end{bmatrix} \in \mathbb{F}^{2560} \quad . \tag{3.4}$$

The set of correction masks is built from a disturbance pattern by rotations and delaying the pattern by 1, 2, ..., 5 words in the same way as described in [33]. In order to find disturbance patterns which can give rise to correction patterns one has to look for bit patterns $b \in \mathbb{F}^{2560}$ that satisfy the following conditions:

C1. the pattern b has to be of the form (3.4), i.e. b is the result of the expansion operation,

C2. the pattern b ends with $5 \cdot 32 = 160$ zero bits (the last five words are zero), because each disturbance is corrected in the next 5 steps, so no disturbance may occur after the word 74,

C3. after delaying a pattern by up to 5 words (that is, shifting bits of b down (right) by $5 \cdot 32 = 160$ positions) the shifted pattern must also be the result of the expansion of its first 512 bits, that is

$$[\underbrace{0 \ldots 0}_{160 \text{ bits}} b_0 \; b_1 \; \ldots \; b_{2399}]^T = E_1([0 \ldots 0 \; b_0 \ldots \; b_{351}]^T) \; .$$

C4. b has both the minimal Hamming weight and the maximal number of non-zero bits in position 1.

Basic Construction

Conditions C1 – C3 imply that in fact we are looking for longer bit sequences of 85 words such that the first 5 words are zero, the next 11 words are chosen in such a way that while the rest of the words are the result of the expansion of the first 16, the last 5 words are zero again. After denoting the first 5 zero words with indices $-5, \ldots, -1$, in positions $0, \ldots, 79$ we get a disturbance pattern which allows for a construction of the corrective pattern.

Using the matrix notation, we are looking for a vector $m \in \mathbb{F}^{512}$ such that $A^4 m$ has 160 trailing zero bits and also $A^{-1} m$ has 160 trailing zeros. As the transformation A is a bijection, this is equivalent to finding a vector

$$v = [v_0, v_1, \ldots, v_{351}, 0, \ldots, 0]^T \in \mathbb{F}^{512} \; ,$$

such that the last 160 bits of $A^{-5}(v)$ contain only zeros, what can be written as

$$\left[\begin{array}{c} x_0 \\ x_1 \\ \vdots \\ x_{351} \\ \hline 0 \\ \vdots \\ 0 \end{array}\right] = \left[\begin{array}{ccccc} a_{0,0} & \dots & \dots & \dots & a_{0,511} \\ & & & & \\ \vdots & & & & \vdots \\ & & & & \\ a_{352,0} & \dots & a_{352,351} & & \\ \vdots & \ddots & \vdots & & \vdots \\ a_{511,0} & \dots & a_{511,351} & \dots & a_{511,511} \end{array}\right] \cdot \left[\begin{array}{c} v_0 \\ v_1 \\ \vdots \\ v_{351} \\ \hline 0 \\ \vdots \\ 0 \end{array}\right] , \tag{3.5}$$

where $A^{-5} = (\, a_{i,j}\,)_{\,0 \le i,j \le 511}$.

This condition means that truncated vectors $\bar{v} = [v_0, v_1, \dots, v_{351}]^T \in \mathbb{F}^{352}$ have to belong to the null-space of the matrix Ω of the form

$$\Omega = \left[\begin{array}{ccc} a_{352,0} & \dots & a_{352,351} \\ \vdots & \ddots & \vdots \\ a_{511,0} & \dots & a_{511,351} \end{array}\right] , \tag{3.6}$$

created as a copy of the lower left part of the matrix A^{-5}. It means that the set of all vectors satisfying properties 1– 2 is a linear subspace of $\mathbb{F}^{2560}$ with elements of the form

$$c = [\, A^{-4}(v)^T \,||\, A^{-3}(v)^T \,||\, A^{-2}(v)^T \,||\, A^{-1}(v)^T \,||\, v^T \,] \ , \tag{3.7}$$

where $v = [\, \bar{v}^T \,||\, 0 \dots 0 \,]^T \in \mathbb{F}^{512}$ and $\bar{v} \in Ker(\Omega)$.

The set of all such vectors c is in fact a linear code C of length $n = 2560$ and, as we have verified that the rank of the matrix Ω is equal to 192, of dimension $k = 192$.

To maximize the probability of a successful correction by the differential pattern, it is necessary to search for the words of minimal Hamming weight and, if possible, for those words with the maximal number of non-zero bits in the position 1.

This is essentially a problem of finding the minimum distance of a linear code, which is known to be NP-hard [149], so there is no easy way of finding optimal

corrective patterns. However, there are a number of probabilistic methods [97, 31] that allow for efficient finding of low-weight codewords in big linear codes.

The second part of the condition C4 can be partially achieved using the fact that the expansion process is invariant with respect to the word rotation. The result of the expansion of 16 input words already rotated by a number of bits is the same as the rotation of the result of the expansion of 16 words performed without rotation. Thanks to that, having a pattern of minimal weight it is easy to transform it to a pattern with the maximal number of ones in the position 1 using the word-wise rotation by an appropriate number of positions. Of course, in general this is the problem of finding codewords with the minimal weighted weight, however, our experiments show that this simplified approach gives very good results.

Reduced variants

The generalization of the construction presented above can be applied to find good differential patterns for reduced versions of SHA-1.

Assume that we want to find a differential pattern for SHA-1 reduced to $16 < s \leq 80$ steps (3.3). Condition C1 implies that the vector $A^{-1}(m)$ has to have 160 trailing zero bits. If we denote the last 160 rows of the matrix A^{-1} as $A^{-1}[352 :: 511]$ then this condition can be written as

$$0 = A^{-1}[352 :: 511] \cdot m \ . \tag{3.8}$$

To formulate a simple description of constraints inferred from condition C2, it is convenient to note that the whole message expansion process can be seen as a linear transform $E_1 : \mathbb{F}^{512} \rightarrow \mathbb{F}^{2560}$ represented by the matrix of the form

$$E_1 = \left[\begin{array}{c} I_{512} \\ \hline A \\ \hline A^2 \\ \hline A^3 \\ \hline A^4 \end{array}\right] \ ,$$

where I_{512} is the identity matrix and A is the linear transform described in Section 3.3.3. Now, if we want to find a differential pattern for s steps, 5 words of the expanded message in positions $s-4$, $s-3$, ..., s have to be zero. In the matrix notation, 160 entries in the vector $E_1 \cdot m$ have to be zero, precisely these in positions $(s-4)\cdot 32$, ..., $s \cdot 32 + 31$. If we denote the matrix created by selecting rows of the matrix E_1 with indices $32(s-4)$, ..., $32s+31$ by $E_1[32(s-4) :: 32s+31]$, then condition C2 can be written as:

$$0 = E_1[32(s-4) :: 32s+31] \cdot m \ . \tag{3.9}$$

Putting together Equations (3.8) and (3.9) we obtain the final result. A message $m \in \mathbb{F}^{512}$ gives rise to the corrective pattern if and only if $m \in Ker(\Psi_s)$, where

$$\Psi_s = \left[\frac{A^{-1}[352 :: 511]}{E_1[32(s-4) :: 32s+31]} \right] \tag{3.10}$$

is a matrix of dimensions 320×512 built by placing rows of $E_1[32(s-4) :: 32s+31]$ below rows of $A^{-1}[352 :: 511]$.

3.3.4 Search for best patterns

We have shown that the problem of finding disturbance patterns with minimal weights can be seen as a problem of finding minimal weight codewords in a linear code. To find them, we use a simplified version of the algorithm by Leon [97] presented in [32]. We use the parameter $p = 3$ to search for all combinations of up to three rows and for each code we apply at least 100 repetitions of the procedure. The results are presented in Table 3.2.

For each variant of SHA-1 (of length 32 - 85) the second column contains the minimal weight of the pattern found. The results marked with (*) are better than those obtained by Biham and Chen [14]. The patterns we investigate are suitable for attacking only last steps of SHA-1. As the first 20 steps of SHA-1 employ the IF Boolean function, the first 16 words of a disturbance pattern cannot have ones in the same bit position in the two consecutive words. Thus for variants longer than 64, we give only lower bounds on the weight of patterns satisfying the IF

Table 3.2: Hamming weights of the best patterns found. The first column contains the number of steps s of a variant. The second column wt contains total Hamming weights of patterns, wt_{20+} – weights of patterns with ignored 20 first steps, column wt_n shows total weights of incomplete patterns for near-collisions (patterns ending with only 4 zero blocks).

s	wt	wt_{20+}	wt_n	s	wt	wt_{20+}	wt_n	s	wt	wt_{20+}	wt_n
32	9	2	9	50	35	14	35	68	> 122	> 78	> 90
33	9	2	9	51	35	15	35	69	> 127	> 81	> 127
34	9	2	9	52	35	16	35	70	> 142	> 80	> 124
35	28	4	24	53	35	16	35	71	> 157	> 94	> 142
36	24	5	24	54	78	36	75	72	> 172	> 93	> 139
37	25	5	25	55	80	39*	73	73	> 139	> 111	> 139
38	30	8	30	56	79	41	72	74	> 139	> 98	> 139
39	39	8*	35	57	72	42	72	75	> 142	> 90	> 142
40	41	11	38	58	73	42	55	76	> 187	> 111	> 187
41	41	12	41	59	91	51	66	77	> 184	> 108	> 184
42	41	13	34	60	66	44	66	78	> 198	> 115	> 177
43	41	17	41	61	66	44	66	79	> 220	> 115	> 220
44	50	15	42	62	66	45	66	80	> 172	> 106	> 172
45	45	15	45	63	107	64	87	81	> 255	> 117	
46	56	23	42	64	> 101	> 60	> 96	82	> 242	> 142	
47	56	24*	35	65	> 113	> 66	> 98	83	> 215	> 163	
48	35	14	35	66	> 98	> 58	> 98	84	> 161	> 101	
49	35	14	35	67	> 127	> 69	> 122	85	> 340	> 177	

condition.

We decided to compute a lower bound because the algorithm we used ensures that there is no codeword of a lower weight with a very high probability. This result is unlikely to be extended in a straightforward way to the case of search for restricted patterns satisfying the IF condition. A way out is finding the lower bound on weights of restricted patterns using unrestricted ones.

According to Biham and Chen [14], it is possible to eliminate the probabilistic behaviour of up to 20 first rounds. Thus the third column (denoted by wt_{20+}) contains minimal weights of patterns where weights of the first 20 steps are not counted.

We are also interested in patterns that do not allow for finding the full collisions but still are suitable for finding near-collisions as this may possibly lead to an easier way of finding multi-block collisions. To obtain them we relax the condition that requires that the last five words must contain zeros only and we allow for non-zero entries in one more block. Weights of the best patterns found this way are listed in the column wt_n.

Table 3.3: A full length difference of weight 44 for unrestricted message expansion of SHA-1

0x00000002	0x00000001	0x00000000	0x00000000	0x00000008
0x00000002	0x00000000	0x00000000	0x00000000	0x00000020
0x00000000	0x00000002	0x00000002	0x00000000	0x00000000
0x00000000	0x00000002	0x00000000	0x00000000	0x00000000
0x00000001	0x00000001	0x00000002	0x00000000	0x00000040
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x80000002	0x00000000	0x00000002	0x00000000	0x00000028
0x00000002	0x00000002	0x00000000	0x00000000	0x00000080
0x80000002	0x00000003	0x00000002	0x00000004	0x00000018
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000002	0x00000002	0x00000000	0x00000000	0x00000100
0x00000000	0x00000002	0x00000000	0x00000008	0x00000020
0x00000003	0x00000000	0x00000000	0x00000000	0x000000a0
0x00000000	0x00000000	0x00000000	0x00000000	0x00000200
0x00000002	0x00000002	0x00000000	0x00000010	0x00000020
0x00000002	0x00000000	0x00000000	0x00000000	0x00000000

Table 3.4: The best differential pattern for the first 34 steps of SHA-1

W[0]= 0x00000002	W[16]= 0x00000000	W[32]= 0x00000000
W[1]= 0x00000000	W[17]= 0x00000000	W[33]= 0x00000000
W[2]= 0x00000002	W[18]= 0x00000000	
W[3]= 0x00000000	W[19]= 0x00000000	
W[4]= 0x00000002	W[20]= 0x00000002	
W[5]= 0x00000000	W[21]= 0x00000000	
W[6]= 0x00000003	W[22]= 0x00000002	
W[7]= 0x00000000	W[23]= 0x00000000	
W[8]= 0x00000000	W[24]= 0x00000000	
W[9]= 0x00000002	W[25]= 0x00000000	
W[10]= 0x00000000	W[26]= 0x00000000	
W[11]= 0x00000000	W[27]= 0x00000000	
W[12]= 0x00000000	W[28]= 0x00000000	
W[13]= 0x00000000	W[29]= 0x00000000	
W[14]= 0x00000002	W[30]= 0x00000000	
W[15]= 0x00000000	W[31]= 0x00000000	

It is interesting to see that the minimal weights we are able to find are growing in quite an irregular fashion. In fact, after a rapid jump after reaching 35 steps and a steady growth up till the step 47, there is an unexpected downfall to the weight 35 in the step 48. The same pattern, presented in Fig. 3.5, is suitable for attacks up to 53 steps. After 53 steps, weights get much higher and as we consider patterns without restrictions imposed by the IF function in the first 20 steps of SHA-1, the best pattern for the full SHA-1 will most likely have weight considerably higher than 172.

However, when we relax all the conditions and look only for patterns that result from the expansion process, we are able to find differences with the weight only 44 for the full length message expansion. Such a difference is presented in Table 3.3.

Table 3.5: The best differential pattern for the last 53 steps of SHA-1

	W[32]=0x00000002	W[48]=0x80000000	W[64]=0x00000002
	W[33]=0x80000000	W[49]=0x00000002	W[65]=0x00000000
	W[34]=0x40000003	W[50]=0x80000001	W[66]=0x00000001
	W[35]=0x00000000	W[51]=0x00000000	W[67]=0x00000000
	W[36]=0x00000001	W[52]=0x00000002	W[68]=0x00000000
	W[37]=0x80000002	W[53]=0x00000002	W[69]=0x00000002
	W[38]=0x80000000	W[54]=0x00000000	W[70]=0x00000000
	W[39]=0x00000002	W[55]=0x00000000	W[71]=0x00000000
	W[40]=0x00000001	W[56]=0x00000002	W[72]=0x00000002
	W[41]=0x00000000	W[57]=0x00000000	W[73]=0x00000000
	W[42]=0x80000002	W[58]=0x00000003	W[74]=0x00000000
W[27]=0x00000000	W[43]=0x00000002	W[59]=0x00000000	W[75]=0x00000000
W[28]=0x00000000	W[44]=0x80000002	W[60]=0x00000002	W[76]=0x00000000
W[29]=0x00000000	W[45]=0x00000000	W[61]=0x00000002	W[77]=0x00000000
W[30]=0x40000000	W[46]=0x80000001	W[62]=0x00000002	W[78]=0x00000000
W[31]=0x00000000	W[47]=0x00000000	W[63]=0x00000000	W[79]=0x00000000

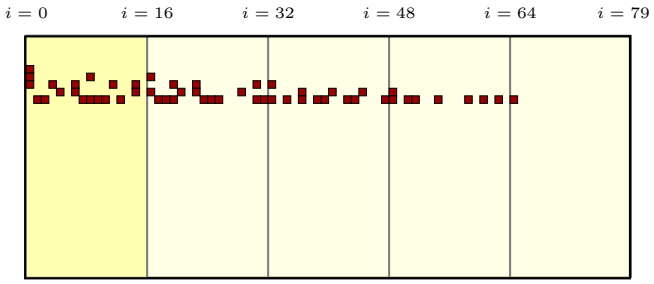


Figure 3.7: Inverse propagation of one bit difference applied in the last segment of SHA-1

3.3.5 Bounds on minimal weights of short patterns

Let us discuss some bounds on minimal weights of corrective patterns. Consider the inverse of the transformation (3.2). It can be written as

$$W_i = W_{i+2} \oplus W_{i+8} \oplus W_{i+13} \oplus ROR^1(W_{i+16}), \quad 0 \leq i < 64 \ , \tag{3.11}$$

where the last 16 words W_{64},...,W_{79} are set arbitrarily.

Although this formula describes essentially the same transformation, if we consider the fact that the rotation is now applied to only one variable distant by 16 steps, the difference propagation of the expansion process described by Equation (3.11) is much worse than the original function. In fact, the difference of one bit in one of the last 16 words generates up to 4 changes positioned 55 to 82 bits, what is illustrated in Fig. 3.7. It is interesting to note that this peculiar behaviour does not depend on the number of positions by which a word is rotated in the algorithm but is rather inherent to the structure of recurrence relations similar to (3.2).

To estimate the minimal number of ones in the expansion process we divide the set of ones in two groups: these in the same position as the initial bit and those in different positions. The size of the first group can be easily found experimentally, as there are only 2^{16} of all bit sequences generated by the following relation

$$w_i = \begin{cases} m_i & \text{for } 0 \leq i < 16, \\ w_{i+2} \oplus w_{i+8} \oplus w_{i+13}, & \text{for } i \geq 16 \end{cases}$$

and much less of them with the first five and the last five elements equal to zero. Minimal weights of such sequences of different lengths are presented in Table 3.6. Note that to estimate the number of ones for a differential pattern of length s, the minimal weight of a sequence of length $s + 5$ has to be considered with 5 leading and 5 trailing zero bits.

Table 3.6: Minimal weights of sequences of length $s + 5$ with 5 leading and 5 trailing zeros generated by the formula $w_i = w_{i+2} \oplus w_{i+8} \oplus w_{i+13}$

s	32–34	35–38	39,40	41	42,43	44–47
min. wt	8	9	11	13	11	14
s	48,49	50	51	52,53	54–56	57–64
min. wt	16	17	16	17	18	19
s	65–67	68–71	72	73–75	76,77	78–85
min. wt	23	22	26	24	29	30

The size of the other group of bits cannot be easily estimated. We only can say that it contains at least one element for sequences longer than 16. This makes our estimation work only for variants that are not too long.

As an example, we can consider the differential pattern for 34 steps. The first set for sequences of length 34 contains at least 8 non-zero bits. The second set must contain at least one bit. Thus, we have shown that the pattern presented in Table 3.4 is the optimal one for that length. This is the same pattern used by Biham and Chen to find collisions for 34 steps of SHA-1 [14].

3.4 Update on SHA-1 attacks

The last two years were extremely fruitful in research and results dedicated to hash functions SHA-0 and SHA-1. In this section we attempt to briefly present results in this area that advance the analysis of SHA designs even further.

3.4.1 In the search for SHA-1 collisions

An analysis similar to ours was performed independently by Rijmen and Oswald [129], they also showed that it is possible to find collisions for SHA-1 reduced to up to 53 steps faster than by birthday paradox. Soon after this, Biham et al. presented an improved search algorithm [15] that finds collisions in SHA-0 using 2^{51} effort, just enough to make it practical and exhibit a colliding pair of messages. They used so-called multi-block differentials, i.e. differential paths that spread through more than just a single instance of the compression function. This allowed for relaxing some conditions at the beginning and at the end of the differential path in each block and decreased the overall complexity of the attack making it practical. However, in a few months the team of X. Wang again surprised the cryptographic community presenting much more efficient attack on SHA-0 [156] and the first theoretical attack on the full SHA-1 [154], both utilising Wang's techniques of modular differentials and message modifications. The crucial difference between previous attempts at SHA-1 and the attack by Wang was a difference in handling the first 20 steps of the function. Instead of following the classical disturbance-corrections strategy that had severe limitations when used in the first round due to paricular behaviour of the Boolean function IF, they opted for a novel approach: finding an irregular differential path through the first 20 steps that matches (or "flows into") the differential path with the lowest possible complexity in the remaining steps. Finding that first part of the differential was again possible thanks to the more flexible method of controlling differentials by the means of both XOR and modular differences. The second part of the differential, used for steps 20-79 was in fact the lowest weight disturbance-corrections differential presented in Table 3.3, also discovered independently by Rijmen and Oswald [129]. However, to achieve full collisions faster than 2^{80} Wang et al. used this technique within multi-block scenario. Combining

those two approaches yields a theoretical attack with total complexity of 2^{69}, later improved to 2^{63}.

Again, this result stimulated a lot of research to better understand Wang's attack and improve the method to bring the complexity of the collision search within the reach of current computing technology. Pramstaller et al. studied the approach that could be seen as extension of [104, 129] where not only the message expansion algorithm, but the whole function is linearised over $\mathbb{F}_2$ and good differential paths are modeled as low-weight codewords in big linear codes. Mendel et al. give a clear summary of Wang's attack in [109] and present a more careful evaluation of the complexity of the attack based on more detailed estimation of probabilities of sufficient conditions.

Wang's approach was also used to attack reduced variants of HAS-160 [162, 35, 106].

Similarly to the case of attacks on MD5, also here one of the most important and interesting questions is how to look for those irregular characteristics that induce the minimal number of approximating conditions and thus have the minimal complexity. Hawkes et al. analysed a method of obtaining such differentials in the first 20 steps of SHA-1 [76]. The problem of finding suitable differentials was considered by Sugita et al. [143]. A recent, significant and beautiful result in this area is due to Ch. De Canniére and Ch. Rechberger [50]. They presented a rigorous analysis of ideas behind Wang et al.'s attacks on SHA-1 and described a general framework for dealing with differential characteristics in SHA-1. They generalised the idea of signed differences even further, to the point where characteristics are basically pairs of expanded messages and register states. This allows for tracking all kinds of differences, in particular those non-linear, irregular ones that are a crucial ingredient of attacks on SHA-1. Having such a rigorous model they developed a method of estimating expected work factor necessary to find a pair of messages that follows given characteristic and this enabled them to derive an automated search algorithm that basically keeps adding conditions in the right order that restrict initially undefined characteristic to obtain a fully determined characteristic that has a reasonable corresponding work factor. Thanks to that approach and many other tricks optimising the whole process, they were able to

get collisions for 64 steps of SHA-1 with work factor of around 2^{35} compression function calls, recently improved to 70 steps with complexity 2^{43} [49]. Another attempt at automated search for differential paths was recently presented by Yajima et al. [157].

3.4.2 Is everything broken?

Not all the work stimulated by Wang's attacks was concerned with cryptanalysis. Some researchers tried to devise ways of countering such attacks within the existing framework of MD-like functions.

Soon after Wang's attacks were published, there was a rather ad hoc proposal to modify the structure of SHA-0 to prevent those kinds of attacks [34]. However, all such structural modifications have the disadvantage of requiring new implementations wherever hash functions are used.

Another direction was so-called message pre-processing proposed by Szydlo and Yin [144], suitable for both MD5 and SHA-1. They observed that to successfully attack those functions, an attacker has to have enough freedom to manipulate messages to obtain the desired characteristic in the first 16 steps and ensure that it will be satisfied in the rest of the function. A simple trick of reducing available message space by fixing the last few words of each message block to zero reduces this freedom dramatically and makes the search for characteristics much more difficult. Since there is no direct control over the propagation of differences through these few steps with fixed message words, differences can mix more thoroughly and it is harder to predict their behaviour at the end of that band. Such a solution is of course temporary but has the big practical advantage of not requiring any modifications to existing MD5 and SHA-1 implementations widely deployed in commercial environments.

An interesting approach, aimed at strenghtening SHA-1 by only slight modifications of the existing structure, was proposed by Jutla and Patthak [85]. They focused on the fact that the set of expanded message words is in fact correspodning to a linear code, just as described in Section 3.3.3, and that the complexity of the attack heavily depends on the weight of the characteristics realising the disturbance-corrections strategy in steps 20–79. This is true also for more ad-

vanced versions of Wang's attack. They asked a question whether by small modifications to the message expansion formula (3.2) it is possible to obtain a code with much higher minimum distance and provably significantly higher weight of words restricted to step 20–79. They proved that the code described by the following recursive fomula

$$W_i = \begin{cases} Y_i \oplus ROL\left(W_{i-1} \oplus W_{i-2} \oplus W_{i-15}\right) & \text{for } 16 \leq i < 36, \\ Y_i \oplus ROL\left(W_{i-1} \oplus W_{i-2} \oplus W_{i-15} \oplus W_{i-20}\right) & \text{for } 36 \leq i \leq 79, \end{cases} \tag{3.12}$$

where $Y_i = W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}$ is the message expansion formula used in SHA-0, gives a code with minimum distance 82 and the minimum weight of the code restricted to steps 20–79 is also 82.

Using similar methods they also proved that the minimum distance of the SHA-1 message expansion code is indeed 44, thus closing the gap between the lower bound and the upped bound obtained previously by experiments [104, 129].

3.5 Summary

Dedicated hash functions of the MD/SHA-1 family ruled the world of practically used cryptographic hash functions for more than 20 years. Despite their relative simplicity, it took many years to develop attacks that are powerful enough to threaten practical applications of such functions.

In this chapter, we studied the extension of the differential attack on SHA-0 to SHA-1. We showed the existence of very low weight differentials in the message expansion algorithm of SHA-1 and used them to develop attacks on reduced variants of SHA-1. Our attacks do not extend to the full function but the same message difference we found was also used by Wang et al. to attack the full function.

Novel ideas of Wang et al. contributed a lot to our understanding of designs based on source-heavy UFNs and opened new avenues of analysing them. It looks like the ability to influence the value of the new word of the state in each step combined with rather weak (in terms of differential behaviour) message expansion

algorithms is the fundamental weakness of designs of that family that can be exploited that way or another.

Right now it seems that trust in such designs has eroded and no new hash function based on design principles of MD would be considered secure unless having some kind of proof of security. Are there any other approaches to designing secure hash functions that are as suitable for high-speed implementations on modern CPUs is an interesting question many are asking right now.

4

Security analysis of the SHA-2 architecture

In August 2002 the National Institue of Standards and Technology announced a new standard FIPS 180-2 [116] that introduced three new cryptographic hash functions, namely SHA-256, SHA-384 and SHA-512. In 2004 the specification was updated with one more hash, SHA-224. All these algorithms are very closely related (in fact SHA-224 is just SHA-256 with truncated hash and SHA-384 is a truncated version of SHA-512) and are called the SHA-2 family of hashes. The design of SHA-512 is very similar to SHA-256, but it uses 64-bit words and some parameters are different to accomodate for this change. Clearly, the fundamental design of this family is SHA-256 and all the other algorithms are variations of that one, so the question of the security of SHA-256 is an extremely interesting one. The latest attacks on other dedicated hash functions, and on SHA-1 in particular, only added weight to that problem.

Comparing to other dedicated hashes, SHA-256 is a much more complex algorithm and its security analysis seems to be a formidable task. Obtaining some partial results for reduced or simplified variants of an algorithm is always a good

first step and we attempted it with SHA-256. This chapter contains results of our investigation into two different simplified variants of SHA-256. After giving the description of SHA-256 we show how to find collisions for a variant with some elements removed. In the second part we present a method of finding collisions for short versions of SHA-256 with all modular additions replaced by XOR operations.

4.1 Description of SHA-256

SHA-256 [116] is an iterated hash function based on the Merkle-Damgård design that uses a compression function mapping 256+512 bits (256 bits of the state and 512 bits of the message) to 256 bits of the new state. To achieve this, the function updates the state of eight 32-bit chaining variables $A, \ldots, H$ according to the values of 16 32-bit words $M_0, \ldots, M_{15}$ of the message. The compression function consists of 64 identical steps presented in Fig. 4.1. The step transformation employs bitwise Boolean functions

$$\begin{aligned} \mathrm{MAJ}(A,B,C) &= (A \wedge B) \vee (A \wedge C) \vee (B \wedge C) \ , \\ \mathrm{IF}(E,F,G) &= (E \wedge F) \vee (\neg E \wedge G) \end{aligned}$$

and two diffusion functions

$$\begin{aligned} \Sigma_0(x) &= ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \ , \\ \Sigma_1(x) &= ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \end{aligned}$$

built from word rotations to the right ($ROTR$) and bitwise XORs denoted by $\oplus$. The i-th step, $i = 0, \ldots, 63$, uses a fixed constant K_i and the i-th word W_i of the expanded message. Constants K_i are defined as the first thirty-two bits of the fractional parts of the cube roots of the first sixty four prime numbers.

The message expansion works as follows. An input message is split into 512-bit message blocks (after padding). A single message block will be denoted either as a row vector $m \in \mathbb{Z}_2^{512}$ or as a vector M of 16 32-bit words $M_t \in \mathbb{Z}_{2^{32}}$, with $0 \leq i < 16$. During the message expansion, this input is expanded into a vector

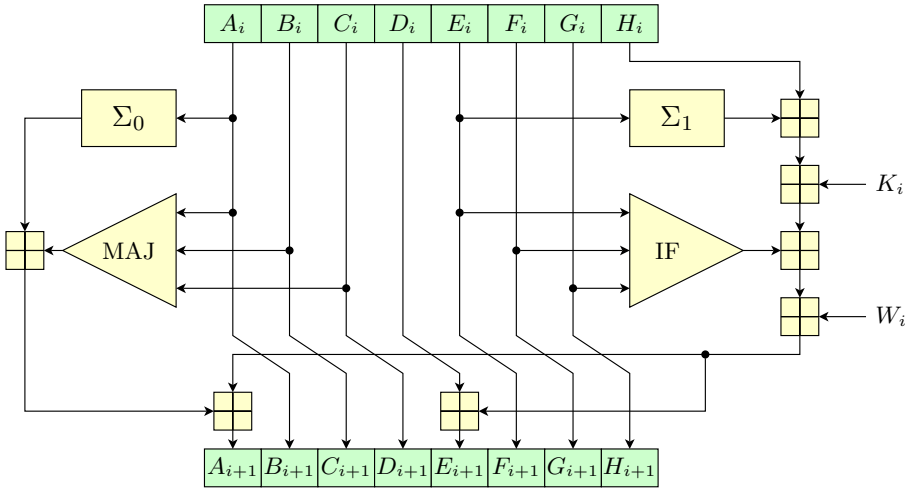


Figure 4.1: One step of the SHA-256 compression function updates the state of eight chaining variables $A, \ldots, H$ using one word W_i of the expanded message.

of 64 32-bit words $W_i \in \mathbb{Z}_{2^{32}}$, which may also be seen as the 2048-bit expanded message row-vector w. The words W_i are generated from the initial message M according to the following formula:

$$W_i = \begin{cases} M_i & \text{for } 0 \leq i < 16 \ , \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} & \text{for } 16 \leq i < N \ . \end{cases} \tag{4.1}$$

This procedure is illustrated in Fig. 4.2. If we set $N = 64$, we get standard SHA-256, taking a different value of N results in a reduced (or extended) variant of it. The functions σ_0 and σ_1 are defined as

$$\sigma_0(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x) \ ,$$
$$\sigma_1(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x) \ ,$$

where $ROTR$ means rotation of bits of the word to the right and SHR denotes shift to the right.

4.2 Attack on a simplified variant of SHA-256

In this section we present an attack on a simplified version of SHA-256 without functions σ_0, σ_1, Σ_1, Σ_1. Firstly, we derive collisions for a fully linearised version and then we show how to extend them to a model with original Boolean functions

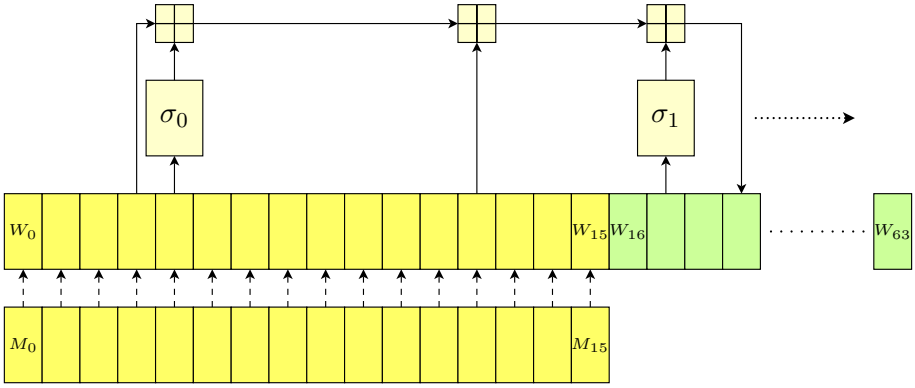


Figure 4.2: The message expansion algorithm of SHA-256 produces 64 words W_0, ..., W_{63} of the expanded message out of 16 initial message words M_0, ..., M_{15} loaded into registers W_0, ..., W_{15}.

in place. The contents of this section is based on the paper [105].

4.2.1 Collisions for a linearised model

We start with investigating an ADD-linear variant of SHA-256, where diffusion boxes are replaced with the identity function,

$$\sigma_0 = \sigma_1 = \Sigma_0 = \Sigma_1 = id \ , \tag{4.2}$$

and Boolean functions are replaced by the addition modulo 2^{32},

$$\mathrm{MAJ}(x, y, z) = \mathrm{IF}(x, y, z) = x + y + z \ . \tag{4.3}$$

This means that the message expansion process has now a simpler, linear form

$$W_i = \begin{cases} M_i & \text{for } 0 \leq i < 16 \ , \\ W_{i-2} + W_{i-7} + W_{i-15} + W_{i-16} & \text{for } 16 \leq i < N \ . \end{cases} \tag{4.4}$$

In fact, the whole function consists only of operations linear with respect to the modular addition. If we introduce a difference $\Delta_i = W'_i - W_i$, we can cancel this disturbance by introducing in the next 8 steps $i+1$, ..., $i+8$ the following sequence of corrections

$$-4\Delta_i, \ 2\Delta_i, \ 2\Delta_i, \ 4\Delta_i, \ 2\Delta_i, \ \Delta_i, \ 0, \ -\Delta_i \ . \tag{4.5}$$

Table 4.1: Correcting a single disturbance Δ_i introduced in step i in an ADD-linearised version of SHA-256

step s	ΔA_s	ΔB_s	ΔC_s	ΔD_s	ΔE_s	ΔF_s	ΔG_s	ΔH_s	ΔW_s
i	0	0	0	0	0	0	0	0	$\mathbf{\Delta}_i$
$i+1$	Δ_i	0	0	0	Δ_i	0	0	0	$-4\Delta_i$
$i+2$	0	Δ_i	0	0	$-2\Delta_i$	Δ_i	0	0	$2\Delta_i$
$i+3$	0	0	Δ_i	0	$-\Delta_i$	$-2\Delta_i$	Δ_i	0	$2\Delta_i$
$i+4$	0	0	0	Δ_i	$-\Delta_i$	$-\Delta_i$	$-2\Delta_i$	Δ_i	$4\Delta_i$
$i+5$	0	0	0	0	Δ_i	$-\Delta_i$	$-\Delta_i$	$-2\Delta_i$	$2\Delta_i$
$i+6$	0	0	0	0	0	Δ_i	$-\Delta_i$	$-\Delta_i$	Δ_i
$i+7$	0	0	0	0	0	0	Δ_i	$-\Delta_i$	0
$i+8$	0	0	0	0	0	0	0	Δ_i	$-\Delta_i$
$i+9$	0	0	0	0	0	0	0	0	

The whole process of correcting a single disturbance is presented in Table 4.1. In the first 4 steps we use corrections that keep differences from influencing register A and later from step $i+4$ we successively cancel differences in the register H.

The next step is to find a disturbance pattern Δ that follows the expansion process and can give raise to a corrective pattern. We will use an argument similar to the one used for finding disturbance patterns for SHA-1 [104, 129, 122]. Let us introduce the necessary notation first. For any vector $s = [s_0, \ldots, s_l]$, let us denote by $\mathrm{Delay}^a(s)$ a vector constructed by preceding elements of s by a zero elements, i.e.

$$\mathrm{Delay}^a(s) = [\underbrace{0, \ldots, 0}_{a \text{ times}}, s_0, \ldots, s_l]$$

and by $\mathrm{Delay}^a_n(s)$ the same vector truncated to only n first elements, i.e.

$$\mathrm{Delay}^a_n(s) = [\underbrace{0, \ldots, 0}_{a \text{ times}}, s_0, \ldots, s_{n-1-a}] \ .$$

Based on this notation, we can state the following simple fact which will be used later on.

Lemma 4.1 *Let $W \in \mathbb{Z}_{2^{32}}^{64}$. If $\mathrm{Delay}^a(W)$ is a result of the expansion using the recursive formula (4.4) with $N = 64+a$, all the vectors $\mathrm{Delay}^b_{64}(W)$ for $0 \leq b \leq a$ are also results of the expansion process (4.4).*

Proof. Each vector $\mathrm{Delay}^b_{64}(W)$ consists of elements of the vector $\mathrm{Delay}^a(W)$ with indices $a-b$, $a-b+1$, $\ldots$, $a-b+63$ and as a part of a sequence following

the recurrence relation, also follows the relation. □

The message expansion can be seen as an ADD-linear transformation $E : \mathbb{Z}_{2^{32}}^{16} \to \mathbb{Z}_{2^{32}}^{64}$. This means that E can be written as a 64×16 matrix

$$E = \begin{bmatrix} I_{16} \\ A \\ A^2 \\ A^3 \end{bmatrix}, \tag{4.6}$$

where I_{16} stands for the identity matrix and A denotes a matrix of the linear transformation producing 16 new words out of 16 old ones according to the recurrence relation (4.4).

The following theorem fully characterises disturbance patterns for an ADD-linear version of SHA-256.

THEOREM 4.2 *Let $\Delta_M = M' - M$ be a message difference. The expanded difference $\Delta = E(\Delta_M)$ is a valid disturbance vector for an ADD–linear variant of SHA-256 if the following conditions are satisfied*

$$\mathbf{0} = A^3[8 :: 16] \cdot \Delta_M \ , \tag{4.7}$$

$$\mathbf{0} = A^{-1}[8 :: 16] \cdot \Delta_M \ , \tag{4.8}$$

where $M[a :: b]$ means a matrix consisting of rows of the matrix M from the a-th row to the b-th row inclusive.

PROOF. The fundamental observation is that each single word Δ_i of the disturbance vector has to be corrected by adding to the next 8 words the following differences defined by Equation (4.5),

$$-4\Delta_i,\ 2\Delta_i,\ 2\Delta_i,\ 4\Delta_i,\ 2\Delta_i,\ \Delta_i,\ 0,\ -\Delta_i \ .$$

This shows that the last non-zero disturbance word may appear in position 55, because we need eight steps 56, ..., 63 to correct it. Thus, the last 8 words of the expanded difference $E \cdot \Delta_M$ have to be zero. Since E is defined by (4.6), this condition can be written as (4.7).

Now, let us consider the following linear combination of Δ and its delayed versions

$$C = \Delta - 4\,\mathrm{Delay}_{64}^{1}(\Delta) + 2\,\mathrm{Delay}_{64}^{2}(\Delta) + 2\,\mathrm{Delay}_{64}^{3}(\Delta) + 4\,\mathrm{Delay}_{64}^{4}(\Delta) + 2\,\mathrm{Delay}_{64}^{5}(\Delta) + \mathrm{Delay}_{64}^{6}(\Delta) - \mathrm{Delay}_{64}^{8}(\Delta)\ . \quad (4.9)$$

It is easy to see that each disturbance word Δ_i in C is corrected by its appropriate multiplicities appearing in the next eight positions and coming from the delayed vectors. Since the message expansion is linear, C is the result of the expansion if and only if all the delayed and truncated vectors $\mathrm{Delay}_{64}^{b}(\Delta)$, $0 \leq b \leq 8$ are results of the expansion process. Lemma 4.1 assures that it is true if $\mathrm{Delay}^{8}(\Delta) = [0,0,0,0,0,0,0,0,\Delta_0,\ldots,\Delta_{63}]^T$ is the result of the (extended, $N = 68$) expansion process. We can achieve this by taking the first 16 words and expanding them forward according to Equation (4.4), but also by taking any 16 consecutive words and expanding partly forward and partly backward. In our case we select elements 8–23 for the expansion. If we index elements of $\mathrm{Delay}^{8}(\Delta)$ starting from -8 and split the vector into two parts: one having negative and the other one having non-negative indices, we can express this requirement equivalently by the following two conditions:

$$[\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,0,0,0,0,0,0,0,0]^T = A^{-1}\cdot\Delta_M \quad \text{and} \quad \Delta = E\cdot\Delta_M\ .$$

Only the first condition, namely that $A^{-1}\Delta_M$ has to end with 8 zeros, has to be satisfied, since Δ is already the result of an expansion. This condition can be written simply as (4.8) what completes the proof. As long as Equations (4.7) and (4.8) are satisfied, Δ is a valid disturbance pattern and C is a complete differential characteristic corresponding to it. □

After obtaining explicit forms of the matrices A^3 and A^{-1} (this is possible since A is a bijection) we solve the system of equations given by (4.7–4.8) over

$\mathbb{Z}_{2^{32}}$ and get the following result:

$$\begin{aligned}\Delta_M = [&\texttt{0x10000000}, \texttt{0xa0000000}, \texttt{0xc0000000}, \texttt{0xa0000000},\\ &\texttt{0xe0000000}, \texttt{0x20000000}, \texttt{0x40000000}, \texttt{0x40000000},\\ &\texttt{0x80000000}, \texttt{0xd0000000}, \texttt{0x10000000}, \texttt{0x60000000},\\ &\texttt{0x50000000}, \texttt{0x40000000}, \texttt{0x70000000}, \texttt{0x30000000}]^T \ .\end{aligned} \tag{4.10}$$

This shows that the solution space is just one-dimensional. Any multiple of Δ_M is also a solution, but since all components of the vector (4.10) have only up to four most significant binary digits different from zero (so they are all of the form $a_i \cdot 2^{28} \pmod{2^{32}}$ where $a_i \in \{0, \ldots, 15\}$, $0 \leq i < 16$), there are only 16 distinct disturbance patterns. Using any of them results in a collision for ADD-linearised SHA-256.

4.2.2 Incorporating Boolean functions

Now let us consider a variant of SHA-256 still without functions σ_0, σ_1, but with both Boolean functions MAJ and IF in place. If we multiply the basic pattern (4.10) by 8 (so shift it left by 3 bit positions), we get a disturbance pattern $\Delta^* = E(8\Delta_M)$ that has non-zero bits at the most significant bits only. The most significant bits of Δ^* are as follows

$$\begin{aligned}&\texttt{1000000001101011 1011100110100110}\\ &\texttt{0000011100101111 1011100000000000}\ .\end{aligned} \tag{4.11}$$

Δ^* is a disturbance pattern that not only follows the message expansion but also allows us to treat it as a binary pattern with a relatively low weight of 27.

We can approximate both Boolean functions with probability at least 1/2 assuming that the function produces an output difference each time the input difference is non-zero. This approximation is shown in Table 4.2.

If we use this approximation and trace how a single bit disturbance Δ_i^* introduced in step i propagates through the next 8 steps, we get the following sequence of corrections:

$$0,\ 0,\ \Delta_i,\ \Delta_i,\ 0,\ 0,\ 0,\ \Delta_i\ , \tag{4.12}$$

which we need in steps $i+1, \ldots, i+8$ in order to cancel the initial disturbance Δ_i. The whole process is very similar to the one used to obtain the sequence of corrections (as given in (4.5)).

A complete differential is obtained in the same way as in the previous case, by adding delayed disturbance patterns multiplied by corresponding coefficients of Equation (4.12), i.e. {0,0,1,1,0,0,0,1}.

This time however, correction process is probabilistic as each active Boolean function almost always (except for input differences $(0,1,1)$ for IF and $(1,1,1)$ for MAJ) introduces a factor of $1/2$. A detailed analysis of these probabilities is presented in Table 4.3. After multiplication of all factors, we obtain a probability for a successful correction equal to 2^{-84}. Further optimisation are also possible as we can choose messages in such a way that conditions for successful correction will be always satisfied for the first 16 rounds, what increases the probability to around 2^{-64}.

This analysis shows that the use of diffusion boxes σ_0, σ_1 and Σ_0, Σ_1 is essential for the security of SHA-256 and also demonstrates that mixing only modular additions with Boolean functions is not enough for constructing a secure hash function.

4.2.3 The Role of diffusion functions

We have shown that the functions Σ_0 and Σ_1 constitute the essential part of the hash function and fulfil two tasks: they add bit diffusion and destroy the

Table 4.2: Probabilities of non-zero output differences for the Boolean functions Ch and Maj

input difference	Ch function		Maj function	
$(\delta_x, \delta_y, \delta_z)$	conditions	prob	conditions	prob
(1,0,0)	$y+z=1$	1/2	$y+z=1$	1/2
(0,1,0)	$x=1$	1/2	$x+z=1$	1/2
(0,0,1)	$x=0$	1/2	$x+y=1$	1/2
(1,1,0)	$x+y+z=0$	1/2	$x+y=0$	1/2
(1,0,1)	$x+y=0$	1/2	$x+z=0$	1/2
(0,1,1)	–	1	$y+z=0$	1/2
(1,1,1)	$y+z=0$	1/2	–	1

Table 4.3: Negative exponents e of the probabilities introduced in step s by Boolean functions MAJ and IF. Columns MAJ and IF show input differences to Boolean functions and 2^{-e} gives probabilities introduced by each step.

s	MAJ	IF	e	s	MAJ	IF	e	s	MAJ	IF	e	s	MAJ	IF	e
0	000	000	0	16	110	010	2	32	011	100	2	48	111	110	1
1	100	100	2	17	111	101	1	33	001	010	2	49	111	011	0
2	010	010	2	18	011	010	2	34	000	001	1	50	011	101	2
3	001	101	2	19	101	001	2	35	000	100	1	51	101	010	2
4	000	110	1	20	110	100	2	36	000	010	1	52	110	101	2
5	000	111	1	21	111	110	1	37	000	001	1	53	111	110	1
6	000	011	0	22	011	011	1	38	100	100	2	54	011	011	1
7	000	001	1	23	001	101	2	39	110	110	2	55	001	101	2
8	000	000	0	24	100	110	2	40	111	011	0	56	000	010	1
9	000	000	0	25	110	011	1	41	011	001	2	57	000	101	1
10	100	100	2	26	011	101	2	42	001	100	2	58	000	010	1
11	110	110	2	27	101	110	2	43	100	110	2	59	000	001	1
12	011	111	2	28	010	011	1	44	010	111	2	60	000	000	0
13	101	111	2	29	001	001	2	45	101	011	1	61	000	000	0
14	010	011	1	30	100	000	1	46	110	001	2	62	000	000	0
15	101	101	2	31	110	000	1	47	111	100	1	63	000	000	0

ADD-linearity of the function.

Good diffusion properties are ensured by a branch number equal to 4 in case of Σ_0 and Σ_1 and equal to three in case of functions σ_0 and σ_1. This means that one-bit input difference will spread to at least three bits at the outputs of Σ_0 and Σ_1 and at least two bits at the outputs of σ_0 and σ_1. This slightly worse result in case of σ_0, σ_1 is due to the single shift used instead of a rotation in those two functions. The reason for that is that such functions prevent the message expansion process from being rotation invariant, because even for a XOR-linearised version, expansion of a message with rotated words is no longer a rotation of the expanded original message as it was the case with SHA-1.

The other crucial property is the degree of non-linearity over $\mathbb{Z}_{2^{32}}$. There are modular differentials for Σ_0 and Σ_1 that hold for one bit input difference e with probability 2^{-3} (necessary for S-boxes used in steps $i+1$, $i+5$) and with probability around 2^{-10} for input difference equal to $\Sigma_0(e)$ (used for Σ_1 in step $i+2$). Using the approach of modular differences it is possible to obtain a corrective pattern for the complete round structure with probability around 2^{-42}. A better result of 2^{-39} was obtained by P. Hawkes et al. [78] by explicit computation of modular differences for Σ_0 and Σ_1, rather than approximating them with a constant differential.

These two properties constitute the foundation of the security of the full SHA-256, as in order to extend this attack and apply corrective patterns in a straightforward way, one would need at least 37 expanded words equal to zero (since at most three corrective patterns can be applied to have the probability of a success larger than 2^{-128}) and this seems unlikely.

4.3 Analysing short versions of SHA-2-XOR

In this section we consider another simplified variant of SHA-256 with all the modular additions replaced by XOR operations. We call this design SHA-2-XOR, following Yoshida et al. [159], who studied such a construction. They showed that it is possible to find pseudo-collisions faster than by birthday paradox for SHA-2-XOR with up to 34 steps using iterative differentials. Here we explore another direction initially inspired by [122] and based on automatic finding of low weight differentials.

4.3.1 Method of the attack

After replacing modular additions + with XOR operations $\oplus$, the only non-linear parts (over $\mathbb{F}_2$) of SHA-2-XOR are the Boolean functions MAJ and IF. Our attack is based on forcing those Boolean functions to behave like linear ones with respect to propagation of differences and it consists of three main stages:

- choose linear approximations of MAJ and IF and construct an $\mathbb{F}_2$–linear model of SHA-2-XOR,
- find a suitable collision-producing difference for the linearised SHA-2-XOR,
- derive a set of conditions under which the real SHA-2-XOR behaves like the linear model with respect to difference propagation
- find a message for which all the conditions are satisfied.

In the rest of this section we will describe the details of each of those steps.

4.3.2 Approximations of MAJ and IF

When considering a linear approximation of a Boolean function for the purpose of this kind of attack one should consider three factors:

1. compatibility of the differential propagation of the function and its linear approximation,
2. impact of the selected approximation on the number of conditions to be satisfied to force the real function to behave like a linear model,
3. probability that the system of approximating conditions corresponding to the differential is consistent,
4. the propagation of differences of the chosen linear approximation.

We explain and discuss all those aspects in detail below.

1. The most important aspect is the selection of such an approximation that is compatible with the differential behaviour of the Boolean function in question. Table 4.4 shows the propagation of $\oplus$-differences for IF and MAJ. It presents the values of differences

$$\Delta\,\mathrm{IF} = \mathrm{IF}(x', y', z') \oplus \mathrm{IF}(x, y, z) \quad \text{and}$$
$$\Delta\,\mathrm{MAJ} = \mathrm{MAJ}(x', y', z') \oplus \mathrm{MAJ}(x, y, z),$$

where $x' = x \oplus \Delta x$, $y' = y \oplus \Delta y$ and $z' = z \oplus \Delta z$. As IF and MAJ are non-linear, their output differences depend not only on the input differences but also on the particular values of x, y and z.

There are two crucial properties to note: for the input difference $(\Delta x, \Delta y, \Delta z) = (0, 1, 1)$ function IF yields the output difference equal to 1 unconditionally. The same is true for the function MAJ and the input difference $(1, 1, 1)$. For that reason if we want to choose a linear approximation of IF, we need to take one with the property that for $(0, 1, 1)$ we have $\Delta f = 1$. The analogous condition has to be true for a function approximating MAJ. Let us call such linear functions *differentially compatible*. The sets of IF and MAJ differentially compatible functions are presented in Table 4.5.

2. Although it is true that for each Boolean function with non-zero input difference exactly one equation involving the inputs to the Boolean function is produced, that does not mean that the total number of linearly independent equations is completely unrelated to the selected approximation of functions MAJ and IF. This is due to the structure of the step transformation. If input differences to the function MAJ in step i are $(\Delta x, \Delta y, \Delta z)$ then in the next step $i+1$ they are $(\Delta w, \Delta x, \Delta y)$. We can exploit that fact to some extent to choose the approximation that reduces the total number of conditions for the differential. If we consider all possible quadruples of input differences $(\Delta w, \Delta x, \Delta y, \Delta z)$ and all possible differentially compatible approximations of MAJ for two consecutive steps, $\Delta MAJ_i(\Delta x, \Delta y, \Delta z)$ and $\Delta MAJ_{i+1}(\Delta w, \Delta x, \Delta y)$, we can see that for two sequences of input differences, namely $(0,1,1,0)$ and $(1,0,0,1)$ there are combinations of approximations of MAJ that yield the same approximation equation and so, effectively, there is only one approximation equation for two steps. They are $(\mathcal{L}_{MAJ_i}, \mathcal{L}_{MAJ_{i+1}}) \in \{(x,y),\ (x,z),\ (y,y),\ (z,x),\ (z, x+y+z),\ (x+y+z, x),\ (x+y+z, x+y+z)\}$. If we take $\Delta MAJ = \Delta y$ or $\Delta MAJ = \Delta x \oplus \Delta y \oplus \Delta z$ we can use those approximations throughout the whole function.

Unfortunately for a cryptanalyst, this small optimisation is not possible for

Table 4.4: Propagation of XOR-differences for Boolean functions IF and MAJ

Δx	Δy	Δz	Δ IF	Δ MAJ
0	0	0	0	0
0	0	1	$x \oplus 1$	$x \oplus y$
0	1	0	x	$x \oplus z$
0	1	1	1	$y \oplus z \oplus 1$
1	0	0	$y \oplus z$	$y \oplus z$
1	0	1	$x \oplus y \oplus z$	$x \oplus z \oplus 1$
1	1	0	$x \oplus y \oplus z \oplus 1$	$x \oplus y \oplus 1$
1	1	1	$y \oplus z \oplus 1$	1

Table 4.5: Linear functions differentially compatible with IF and MAJ

Boolean function	differentially compatible linear functions
IF	y, z, $x \oplus y$, $x \oplus z$
MAJ	x, y, z, $x \oplus y \oplus z$

the Boolean function IF.

3. The choice of the approximation have also impact on the probability that the system of approximating equations is consistent. The detailed analysis is presented in appendix A, here we only summarise the results. We performed experiments with systems of approximating equations corresponding to randomly generated differentials of different lengths. For each differentially compatible linear approximation of MAJ and IF we generated at least 100000 of random differentials of each length 10, 20, ..., 60 and computed the fraction of consistent systems. The results show that we have the best chance of getting a consistent system when we use z as the approximation of IF and $x \oplus y \oplus z$ as the approximation of MAJ with y being only a slightly worse option for MAJ.

4. We can expect that the choice of a linear approximation may influence the complexity of differentials. Intuitively, the worse avalanche effect of the linear function, the less complex differentials we expect to get as there will be less "mixing" introduced by linear approximations. From that point of view, $x \oplus y$ and $x \oplus z$ are less attractive approximations of IF than y and z. Similarly, for MAJ it is better to choose x, y or z rather than $x \oplus y \oplus z$.

Once we decide on the linear approximations, we can replace Boolean functions IF and MAJ with them. The resulting function is $\mathbb{F}_2$-affine and if we consider the propagation of differences, it is $\mathbb{F}_2$-linear. We will call a function with all Boolean functions IF and MAJ replaced by $f(x, y, z) = z$ and with all constants K_i set to zero SHA-2-XOR-Lin.

4.3.3 Finding collision-producing differentials

The next step is to find optimal collision-producing differentials for SHA-2-XOR-Lin. Note that SHA-2-XOR-Lin can be seen as a function $SXL : \mathbb{F}_2^{256} \times \mathbb{F}_2^{512} \to \mathbb{F}_2^{256}$ linear over $\mathbb{F}_2$ and so it has its matrix form which is a matrix over $\mathbb{F}_2$ of dimensions 768×256.

Now, every bit string $(\Delta_{IV}, \Delta_M) \in \mathbb{F}_2^{256} \times \mathbb{F}_2^{512}$ such that

$$SXL(\Delta_{IV}, \Delta_M) = 0$$

is a pseudo-collision-producing difference for SHA-2-XOR-Lin (as well as for the affine version with constants K_i in place). In other words, the set of all differences resulting in a pseudo-collision is the kernel of the linear map SXL,

$$K_P = \mathrm{Ker}(SXL). \tag{4.13}$$

As K_P forms a linear subspace, it has a basis and we will denote its matrix as $\mathcal{B}(K_P)$. Every difference that leads to a pseudo-collision is a certain combination of rows of the matrix $\mathcal{B}(K_P)$. It is easy to verify experimentally that the dimension of K_P is 512 and so the dimensions of the matrix $\mathcal{B}(K_P)$ are 512×768.

If we are interested in real collisions, we need to consider the function $SXL_M : \mathbb{F}_2^{512} \to \mathbb{F}_2^{256}$ which is the function SXL with $\Delta_{IV} = 0$,

$$SXL_M(\Delta_M) = SXL(0, \Delta_M) \quad \forall\, \Delta_M \in \mathbb{F}_2^{512}. \tag{4.14}$$

Now the set of all collision-producing differences is the kernel of the function SXL_M :

$$K_C = \mathrm{Ker}(SXL_M)\ . \tag{4.15}$$

Again, it will be more convenient for us later to operate on a basis matrix of this linear subspace, $\mathcal{B}(K_C)$. This time it is a matrix of dimensions 256×512.

We need to identify now what is the measure of the quality of the difference. It is clear that we are looking for message differences that yield a small number of different bits in chaining variables $A_1,\ldots,H_1, \ldots, A_N,\ldots, H_N$, as each set of non-zero input differences to a Boolean function results in a condition that has to be satisfied later. The less conditions, the more feasible is the attack. To state our minimisation problem more precisely, we need the following observations.

The values of the registers $A_i,\ldots,H_i$ can be expressed as (linear) functions $A_i(x_{IV}, x_M), \ldots, H_i(x_{IV}, x_M)$, $i \in \{1..N\}$ of the input (x_{IV}, x_M).

Our goal is to minimise the sum of Hamming weights of all the registers in

all steps from 1 to N, i.e. the value of

$$Wt(k_{IV}, k_M) = \sum_{i=1}^{N} \big(wt(A_i(k_{IV,k_M})) + wt(B_i(k_{IV,k_M})) + \cdots + wt(H_i(k_{IV,k_M})) \big) \tag{4.16}$$

over all possible values of $(k_{IV}, k_M) \in K_P$ (or $(0, k)$, where $k \in K_C$, if we are looking for collisions) as these values will give us pseudo or real collisions at the end.

An important observation that makes our problem more feasible is that the values of B_{i+1}, C_{i+2}, D_{i+3} for $1 \leq i \leq N-3$ are the same as the value of A_i. The same is with the values of F_{i+1}, G_{i+2}, H_{i+3} which are equal to E_i. Using that fact we can estimate the value of the sum (4.16) by

$$Wt^*(k_{IV}, k_M) = 4 \sum_{i=1}^{N} \big(wt(A_i(k_{IV,k_M})) + wt(E_i(k_{IV,k_M})) \big) \tag{4.17}$$

There is no equality between these two values in general, as the values of B, C, D, F, G, H in the first three steps come from the initial values rather than from the values of A and E and in the last three steps the new values of A and E are repeated less than three times. However, (4.17) gives a good estimation of (4.16) and for one-block collisions they are equal.

Let us introduce a function $\Psi_{P,N} : \mathbb{F}_2^{256} \times \mathbb{F}_2^{512} \to (\mathbb{F}_2^{32} \times \mathbb{F}_2^{32})^N$ that for a bit vector of input differences (k_{IV}, k_M) returns a vector of differences in chaining variables A and E in steps 1 to N. More formally,

$$\Psi_{P,N}(k_{IV}, k_M) = (A_1(k_{IV}, k_M), E_1(k_{IV}, k_M), \ldots, A_N(k_{IV}, k_M), E_N(k_{IV}, k_M)) \\ \text{for all } (k_{IV}, k_M) \in \mathbb{F}_2^{256} \times \mathbb{F}_2^{512}. \tag{4.18}$$

Since $\Psi_{P,N}$ is an $\mathbb{F}_2$-linear function, we we can think of $\Psi_{P,N}$ as a matrix of dimensions $768 \times 64N$. After multiplying a row vector of length 768 representing initial state and message difference, we get a vector of length $64N$ containing bit differences of the registers A and E. Now, if we multiply the matrix $\mathcal{B}(K_P)$ by the matrix $\Psi_{P,N}$ we get a matrix of dimensions $512 \times 64N$. Let us call this matrix $R_{P,N}$,

$$R_{P,N} = \mathcal{B}(K_P) \cdot \Psi_{P,N}. \tag{4.19}$$

To minimise the value of (4.17), we need to find a linear combination of rows of $R_{P,N}$ with minimal Hamming weight. This problem is precisely the problem of finding minimum weight codewords in a linear code with the generating matrix $R_{P,N}$.

For the case of collisions, we define analogously $\Psi_{C,N} : \mathbb{F}_2^{512} \to (\mathbb{F}_2^{32} \times \mathbb{F}_2^{32})^N$ such that

$$\Psi_{C,N}(k_M) = (A_1(0, k_M), E_1(0, k_M), \ldots, A_N(0, k_M), E_N(0, k_M)), \quad k_M \in \mathbb{F}_2^{512}. \tag{4.20}$$

The generating matrix describing the code has now the form

$$R_{C,N} = \mathcal{B}(K_C) \cdot \Psi_{C,N}, \tag{4.21}$$

and is of dimensions $256 \times 64N$.

Experimental results The dimensions and sizes of the codes in question (namely $R_{P,N}$ and $R_{C,N}$) place them far beyond the reach of standard probabilistic techniques for finding low-weight codewords. Essentially, the security of SHA-2-XOR is based on the infeasibility of finding such low-weight differences. There is no proof that they do not exist but only a strong experimental evidence that at least finding them seems to be very difficult.

Our experiments were focused on quite short variants, with $N = 20$, $N = 24$, $N = 28$ and $N = 32$, to test the usefulness of this approach. However, even such short variants are interesting because the best result so far is the proof that pseudo-collisions can be found with complexity 2^{120} for a variant with $N = 34$ steps [159].

A sample collision-producing characteristics for $N = 24$ is presented in Table. 4.6.

Experimental results show that finding low-weight codewords in unrestricted codes doesn't yield satisfying results as the complexity of search increases exponentially with the dimension of the code. Thus we decided to search for useful characteristics in subcodes for which we forced some positions to be zero. The comparison of results for a short variant of 20 steps is presented in Table 4.7.

4.3.4 Deriving the set of conditions

Once we have a low weight differential characteristics we can proceed to derive the set of conditions that guarantee the propagation of differences through Boolean functions IF and MAJ according to that characteristics. That way we can significantly improve the probability that the message difference for the real function will propagate in the expected way. If we decided to use a function $f_1(x, y, z)$ as differential approximation of IF and $f_2(x, y, z)$ as an approximation of MAJ, we need conditions that ensure that $\Delta f_1 = \Delta\,\mathrm{IF}$ and $\Delta f_2 = \Delta\,\mathrm{MAJ}$. For our case where $f_1 = f_2 = z$ those conditions are listed in Table 4.8. Using the fact that $B_{i,j} = A_{i-1,j}$, $C_{i,j} = A_{i-2,j}$ and $F_{i,j} = E_{i-1,j}$, $G_{i,j} = E_{i-2,j}$ we can easily rewrite these conditions in terms of bits of A and E only.

If the differential characteristic is not very sparse we can expect the number of equations to be considerably smaller than the weight of the differential as in some cases two or three bits will contribute to only one condition. In practice, for the differential presented in Table 4.6 of weight 1020, we got 451 equations in total. For 20-step characteristics of total weight 572 we obtained 298 independent equations.

The crucial problem is that to be able to find a pair of messages conforming to the characteristics, two sets of equations: one involving conditions on bits

Table 4.6: Sample (dense) collision producing differential for 24 steps of SHA-2-XOR-Lin. The total weight of this path is 1020 but the corresponding system has 451 conditions.

s	A	B	C	D	E	F	G	H	W
0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000002
01	00000002	00000000	00000000	00000000	00000002	00000000	00000000	00000000	88d00900
02	00800000	00000002	00000000	00000000	80900800	00000002	00000000	00000000	4c365267
03	06000404	00800000	00000002	00000000	06200006	80900800	00000002	00000000	3c6cf61a
04	15049000	06000404	00800000	00000002	34b4b11b	06200006	80900800	00000002	1a3d4c7e
05	18c001c4	15049000	06000404	00800000	8f418db6	34b4b11b	06200006	80900800	64aeef50
06	20230000	18c001c4	15049000	06000404	2eb4d216	8f418db6	34b4b11b	06200006	56bfb089
07	b5305ec9	20230000	18c001c4	15049000	223d0b55	2eb4d216	8f418db6	34b4b11b	cd33ac2c
08	14d510bc	b5305ec9	20230000	18c001c4	436b199c	223d0b55	2eb4d216	8f418db6	ab6146c0
09	09290490	14d510bc	b5305ec9	20230000	e55d1780	436b199c	223d0b55	2eb4d216	c560e84c
10	a3033800	09290490	14d510bc	b5305ec9	b4c82e81	e55d1780	436b199c	223d0b55	22c447b4
11	01906831	a3033800	09290490	14d510bc	4450f2d1	b4c82e81	e55d1780	436b199c	d8137972
12	00000000	01906831	a3033800	09290490	5db0c6a5	4450f2d1	b4c82e81	e55d1780	6b2826ad
13	00000000	00000000	01906831	a3033800	aa2a3c90	5db0c6a5	4450f2d1	b4c82e81	34ab1181
14	00000000	00000000	00000000	01906831	a2935031	aa2a3c90	5db0c6a5	4450f2d1	90f6338f
15	00000000	00000000	00000000	00000000	01906831	a2935031	aa2a3c90	5db0c6a5	fd889118
16	00000000	00000000	00000000	00000000	00000000	01906831	a2935031	aa2a3c90	08b96ca1
17	00000000	00000000	00000000	00000000	00000000	00000000	01906831	a2935031	a3033800
18	00000000	00000000	00000000	00000000	00000000	00000000	00000000	01906831	01906831
19	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
20	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
21	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
22	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
23	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
24	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	

Table 4.7: Bounds on weights of different restricted subcodes for 20-step collisions for SHA-2-XOR with approximations $\Delta MAJ = \Delta y$ and $\Delta IF = \Delta y$.

positions forced to zero	dimension	lower bound	upper bound
-	256	15	187
[739..1280]	128	23	141
[641..1280]	64	46	142
[513..1280]	2	194	194
$[1..32] \cup [641..1280]$	32	144	144
[577..1280]	32	126	143
[617..1280]	40	89	140
[621..1280]	44	73	139
[625..1280]	48	71	143

Table 4.8: Approximating conditions for $\Delta z = \Delta\,\mathrm{IF}$ and $\Delta z = \Delta\,\mathrm{MAJ}$

$(\Delta A_{i,j}, \Delta B_{i,j}, \Delta C_{i,j})$ $(\Delta E_{i,j}, \Delta F_{i,j}, \Delta G_{i,j})$	condition for IF	condition for MAJ
(0, 0, 0)	-	-
(0, 0, 1)	$A_{i,j} = 0$	$E_{i,j} \oplus F_{i,j} = 1$
(0, 1, 0)	$A_{i,j} = 0$	$E_{i,j} \oplus G_{i,j} = 0$
(0, 1, 1)	-	$F_{i,j} \oplus G_{i,j} = 0$
(1, 0, 0)	$B_{i,j} \oplus C_{i,j} = 0$	$F_{i,j} \oplus G_{i,j} = 1$
(1, 0, 1)	$A_{i,j} \oplus B_{i,j} \oplus C_{i,j} = 1$	$E_{i,j} \oplus G_{i,j} = 0$
(1, 1, 0)	$A_{i,j} \oplus B_{i,j} \oplus C_{i,j} = 1$	$E_{i,j} \oplus F_{i,j} = 1$
(1, 1, 1)	$B_{i,j} \oplus C_{i,j} = 0$	-

$A_{i,j}$ and another involving all $E_{i,j}$ have to be simultaneously consistent. The probability of obtaining such systems is considered in the next section in details.

If we have a differential for which both systems are consistent, we can try to find a solution i.e. a message that satisfies all the approximation equations.

4.3.5 Satisfiability of systems of approximating conditions

The important question now is what is the probability that a system obtained from a differential pattern is consistent. It is easy to see that a system of constraints on bits of registers A_i can be separated into 32 independent systems, each of them consisting of variables $A_{i,j}$, $i = 1..N$ only, i.e. variables corresponding to only j-th bit of words A_i, $i = 1..N$. For example, for the approximation $\Delta IF(x, y, z) = \Delta z$ there are only three types of equations involving registers A and they are of the form

$$0 = 0 \tag{4.22a}$$

$$x_i = 0 \tag{4.22b}$$

$$x_{i-2} \oplus x_{i-1} = 0 \tag{4.22c}$$

$$x_{i-2} \oplus x_{i-1} \oplus x_i = 1 \tag{4.22d}$$

for $i = 1..N$.

One of the smallest inconsistent systems for this approximation is presented below

$$x_1 = 0$$

$$\textit{... any equation ...}$$

$$x_1 + x_2 + x_3 = 1$$

$$x_2 + x_3 = 0$$

If a bigger system contains this pattern of equations anywhere, it is inconsistent.

This suggests two natural questions. The first one is what is the probability that a system obtained from a differential pattern is consistent. The second one asks whether the probability depends on the approximation of the Boolean

Table 4.9: Probability that a system of conditions corresponding to a random differential for N steps is consistent depends on the selected approximation of the Boolean function.

	Approximation of IF				Approximation of MAJ			
N	y	z	$x \oplus y$	$x \oplus z$	x	y	z	$x \oplus y \oplus z$
10	0.905	0.918	0.907	0.847	0.621	0.912	0.622	0.934
20	0.773	0.826	0.791	0.707	0.356	0.812	0.356	0.859
30	0.661	0.738	0.686	0.587	0.208	0.717	0.208	0.790
40	0.565	0.663	0.598	0.489	0.118	0.638	0.120	0.723
50	0.483	0.598	0.520	0.407	0.068	0.565	0.068	0.664
60	0.417	0.534	0.453	0.344	0.040	0.502	0.039	0.609

function we chose.

To answer those questions, we conducted some experiments with randomly generated systems consisting of approximating equations for all possible differentially compatible linear approximations of Boolean functions MAJ and IF. The results are summarised in Table 4.9. It should be noted that the probabilities given in tables correspond to systems involving only one bit position of registers A or E. If we had a random system of approximating equations the final probability would be obtained by raising the result to the power 32, as there are 32 bits in each register.

Those results may seem pessimistic as it would be practically impossible to get a consistent system for any interesting variants of SHA-2-XOR. However, there is a simple trick that somehow improves the situation. It is enough to generate differentials that have bands of zeros interleaving parts with difference bits. That way we split the system of equations into smaller, independent ones and decrease the probability of occurrence of inconsistent patterns.

Once we have a system that is consistent, we can proceed to the last step of the attack that tries to find a message that generates the state of the function for which all the equations are satisfied.

4.3.6 Finding a message satisfying approximating conditions

The task of finding a message that generates the state of the hash function that satisfies all the approximating conditions is quite difficult in general – we are looking for a solution of a large system of quadratic equations over $GF(2)$.

$$
\begin{array}{lll}
E_{8,2} = E_{7,2} & E_{8,13} = 1 & E_{8,23} = 1 \\
A_{8,2} = A_{7,2} & E_{8,15} = 0 & E_{8,26} = E_{7,26} + 1 \\
E_{8,3} = 1 & A_{8,16} = A_{0,16} & A_{8,26} = A_{4,26} \\
A_{8,4} = A_{4,4} & E_{8,22} = 1 & E_{8,28} = 1
\end{array}
$$

Figure 4.3: An example of a system of approximating equations involving variables up to step 8, i.e. taken from the set $\mathcal{E}[8]$.

However, in the first 16 steps we have complete freedom of selecting the values of message words and that way we can control the values of registers A_i and E_i for $i = 1, \ldots, 16$ to a certain extent.

The general idea of our heuristic algorithm is to incrementally make all equations satisfied advancing step by step in the structure, correcting bits with wrong values by flipping appropriate bits of message words while making sure the changes do not spoil equations that are already statisfied.

Consider the set of approximating equations, let us call it $\mathcal{E}$. We can partition it into a family of sets, each one containing equations involving variables up to step k only. If we denote such sets as $\mathcal{E}[k]$ we have $\mathcal{E} = \sum_k \mathcal{E}[k]$. All the equations can be rewritten in such a way that the variable from the latest step is on the left-hand side and all the other variables (and possibly a constant) are on the right-hand side, as illustrated in Fig. 4.3. It is clear that if we can set the values of variables on the left-hand side, i.e. change the values of registers A and E in the last step, we can always satisfy such equations (provided that the system is consistent).

Let us analyse now how we can change the values of bits of A_i and E_i by manipulating values of W_{i-1} and W_{i-2}. There are two propagation rules that are useful for us.

Rule #1: Direct modification. In case of W_{i-1} the rule is simple. Flipping the bit $W_{i-1,b}$ flips both $A_{i,b}$ and $E_{i,b}$.

Rule #2: Indirect modification. This rule describes the impact of flipping the bit $W_{i-2,b}$ on registers A_i and E_i. Here the situtation is more complex and is best explained by a picture presented in Fig. 4.4. Bit $W_{i-2,b}$ changes bits $A_{i-1\,,b}$ and $E_{i-1\,,b}$. A change in $A_{i-1\,,b}$ propagates through Σ_0 and reverses bits

$A_{i,b-2 \bmod 32}$, $A_{i,b-13 \bmod 32}$ and $A_{i,b-22 \bmod 32}$. Similarly, a change in $E_{i-1,b}$ propagates through Σ_1 to simultaneously flip bits of A_i and E_i with indices $b-6 \mod 32$, $b-11 \mod 32$ and $b-25 \mod 32$. Additionally, depending on the values of $B_{i-1,b}$, $C_{i-1,b}$ and $F_{i-1,b}$, $G_{i-1,b}$, the difference may propagate through Boolean functions MAJ and IF respectively, influencing bits $A_{i,b}$ and $E_{i,b}$. More formally, if we assume that there are no differences in registers A_{i-2},

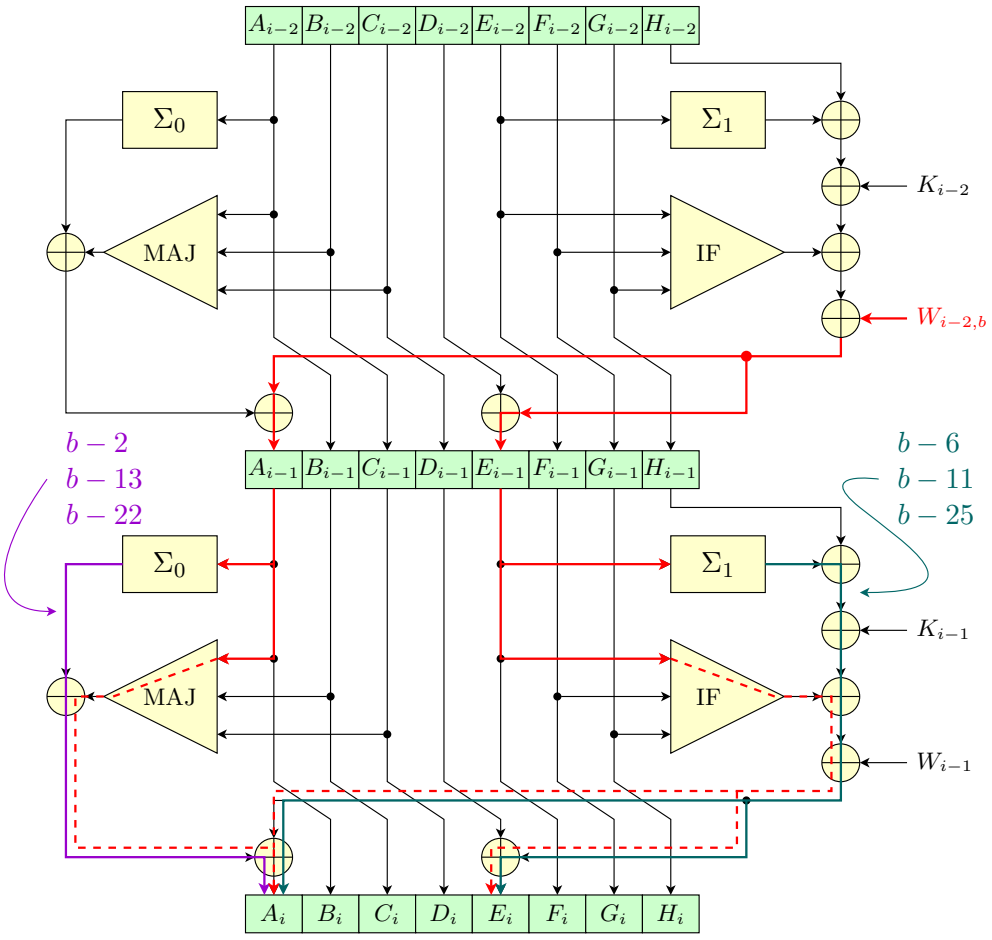


Figure 4.4: Propagation of differences throughout two steps of SHA-2-XOR when one bit of W_{i-2} is changed.

$\ldots, H_{i-2}$ we can describe the differences in A_i and E_i as follows

$$\begin{aligned}\Delta A_i =& \Delta W_{i-1} \oplus \Sigma_0(\Delta W_{i-2}) \oplus \Sigma_1(\Delta W_{i-2}) \oplus \mathrm{MAJ}(\Delta W_{i-2}, B_{i-1}, C_{i-1}) \\ & \oplus \mathrm{IF}(\Delta W_{i-2}, F_{i-1}, G_{i-1}) \ , \\ \Delta E_i =& \Delta W_{i-1} \oplus \Sigma_1(\Delta W_{i-2}) \oplus \mathrm{IF}(\Delta W_{i-2}, F_{i-1}, G_{i-1}) \ .\end{aligned}$$

Observe that if the values of registers B_{i-1}, C_{i-1} and F_{i-1}, G_{i-1} are fixed, the propagation of differences through Boolean functions is in fact described by the fifth line of Table 4.4 since we have non-zero input difference in the first input and zero differences on the second and third input. For a non-zero input difference in j-th bit of A_{i-1} the ouptut difference is equal to the value of $B_{i-1,j} \oplus C_{i-1,j}$ in case of MAJ and $F_{i-1\,,j} \oplus G_{i-1\,,j}$ in case of IF. This means that we can model both MAJ and IF as linear functions, parametrised by fixed values of the other registers. In turns, it follows that the whole function $Flip : \mathbb{F}_2^{32} \times \mathbb{F}_2^{32} \to \mathbb{F}_2^{32} \times \mathbb{F}_2^{32}$ that takes differences in words W_{i-2} and W_{i-1} and returns differences in registers A_i and E_i is in fact linear over $\mathbb{F}_2$. It means that for any desired pattern of output differences (i.e. specific changes to bits of A_i and E_i) we can easily check if there exist input differences (i.e. differences in W_{i-2} and W_{i-1}) that yield this output difference and we can find sets of such inputs efficiently.

This constitutes the foundation of our algorithm: we want to change some bits of registers A_i and E_i that take part in equations from $\mathcal{E}[i]$ not yet satisfied while keeping differences equal to zero for those bit positions that are present in already satisfied equations. That way, if only the solution exists, we can find it and correct all the equations in the step at the same time.

There is one more important thing to note. When going from step $i-1$ to step i we do not want to spoil equations from $\mathcal{E}[i-1]$ involving bits of A_{i-1} and E_{i-1}. Some of the bits of these registers have to remain unchanged and thus we cannot use all possible bits of W_{i-2} because that could spoil some earlier equations. Also, if we have an equation of the form $A_{i,b} = A_{i-1,b}$ and, because of the behaviour of MAJ and IF, flipping the bit of $W_{i-2,b}$ would flip both bits $A_{i-1\,,b}$ and $A_{i\,,b}$ we have to mark the bit $A_{i-1\,,b}$ as fixed to either force the change of $A_{i\,,b}$ by using other bits of W_{i-2} if the equation is not satisfied or preserve the value of the register if the equation is satisfied.

Let us describe now how the algorithm advances one step, going from the state where all equations are satisfied up to step $i-1$ to the next state where all equations up to step i are satisfied.

We assume that all equations from the set $\sum_{k=0}^{i-1} \mathcal{E}[k]$ are satisfied and that all bits in registers A, E up to step $i-2$ are fixed, i.e. we do not change them

and there are no differences there. Also, as we mentioned before, some of the bits of registers A_{i-1} and E_{i-1} that appear in approximating equations for step $i-1$ are also fixed and we cannot modify them. Let us denote the set of indices of bits of A fixed in step k by $FixedA_k$. Similarly, we use the notation $FixedE_k$ for the set of indices of bits of register E_k that are fixed. The algorithm proceeds as follows.

- Looking at the left-hand sides of equations from set $\mathcal{E}[i-1]$ determine which bits of A_{i-1}, E_{i-1} have to be fixed and compute sets $FixedA_{i-1}$ and $FixedE_{i-1}$.

- Assign random values to bits of W_{i-1} that do not correspond to fixed positions in registers A_{i-1} and E_{i-1}, i.e. bits with indices in $\{0,\ldots,31\} \setminus (FixedA_{i-1} \cup FixedE_{i-1})$.

- Determine the linear transformation $Flip$ and its matrix.

- Looking at the set $\mathcal{E}[i]$ find those equations that inolve bits at the same position in step $i-1$ and i (e.g. equations like $A_{i,k} = A_{i-1,k}$ or $E_{i,k} = E_{i-1,k} + E_{i-2,k} + 1$). If the function $Flip$ modifies both bit positions, add indices corresponding to those bits to $FixedA_{i-1}$ and $FixedE_{i-1}$.

- Create matrix $Flip'$ by selecting columns of $Flip$ with indices that do not correspond to fixed bits of A_{i-1} and E_{i-1}. We assume that bits of A_{i-1} correspond to columns $0,\ldots,31$ and bits of E_{i-1} correspond to columns $32,\ldots,63$.

- Create matrix Q by picking only those rows of the matrix $Flip'$ that correspond to bits of A_i and E_i that appear in left-hand sides of equations from $\mathcal{E}[i]$. Matrix Q describes now how by flipping only allowed bits of W_{i-2} we can change the values of bits that appear in target variables of equations of step i.

- Create a vector y of length $|\mathcal{E}[i]|$ that contains zeros in positions corresponding to equations from $\mathcal{E}[i]$ that are satisfied and ones in positions corresponding to equations that need to be corrected.

- Look for a solution x of the matrix equation

$$y = Q \cdot x \ . \tag{4.23}$$

  If a solution x exists, recreate the corresponding values of ΔW_{i-2}, apply the correction to the word W_{i-2} and return success. If there is no solution, return failure.

This method of satisfying equations in a single step can be used to obtain a simple algorithm that works for all $1, \ldots, 16$. We start with step 1. For equations from $\mathcal{E}[0]$ and $\mathcal{E}[1]$ we may need to correct some bits of the initial state, since there are no corresponding message words that can be used by our method to modify registers A_0, E_0 and A_1, E_1. Starting from step 2, we use the algorithm described above. If the algorithm successfully goes through all steps, we are done. If it fails in a step, one of the possible solutions is to go back one step and hope that a fresh pseudorandom value of W_{i-2} will be sufficient to pass the step next time. Unfortunately, this method is only a heuristics as it is not guaranteed to terminate. This poses a significant problem for dense characteristics that require a lot of conditions to be satisfied.

We can estimate however, when we can hope for a solution. Clearly, to get a solution of the system (4.23) with high probability, the number of rows should be smaller than the number of columns of Q. This means that the number of bits we want to fix in step i should be less than the number of free bits remaining in step $i-1$, a simple necessary condition that allows to asses quickly the potential of a differential path. Of course, the chance of getting a solution depends also on the form of the matrix $Flip$ but it seems difficult to derive concrete conditions here.

4.3.7 Discussion and related results

In this section we explored a possible method of obtaining collisions for reduced variants of SHA-2-XOR. Even though it seems to be an interesting line of research, so far it has some serious limitations that have to be overcome to make it more practical. The first problem is finding linear characteristics that are of suffi-

ciently low weight. In general, the smaller the weight, the better the probability. Unfortunately, as we have already mentioned, this problem is NP-hard and we can only hope to push boundaries of what is feasible in practice. Considering that the attack on a full function would require codes of length around 4096 and dimensions a few hundreds, one cannot expect to find characteristics of very low weights. Still, computational advances in the area of coding theory may extend the potential of this attack. Note that it is enough to find a characteristics only once, and growing popularity and power of distributed and grid computing may be a tool to achieve this.

However, what in fact really influences the probability is the number of equations that have to be satisfied. Table 4.6 shows that for dense characteristics there is a substantial difference between the weight of the characteristics and the number of approximating conditions that have to be satisfied. This direction could be investigated further.

Another issue is the problem of finding messages that satisfy the system of conditions derived from the chosen approximation of Boolean functions past the first 16 steps. It seems that the main obstacle is that there is no simple way of progressively satisfying equations after step 16 since changes that are required to fix new equations may spoil conditions already satisfied. It may be possible to use the method of neutral bits [13], but different message expansion algorithm certainly makes this more difficult.

Finally, the study of SHA-2-XOR is interesting mainly because of its close relationship with the original design SHA-256. However, going back to the original design by straightforward restoring modular additions while still preserving the differential seems to be impossible because of a large number of additional conditions that must be imposed on values of registers to force ADDs to behave like XORs.

In spite of all the aforementioned problems, it seems that there are still many unexplored directions related to this method and this approach may deserve more attention.

Related results A similar analysis, focused on reduced variants of the original function SHA-256, was performed by Mendel et al. and published in [108]. The

two approaches have common principles, but the results of Mendel et al. seem to be considerably better due to a few important differences. The first difference is the choice of the approximation. In our analysis, we considered only differentially compatible approximations but it seems that the best choice is the approximation by the constant function. It results in characteristics with much lower densities and this greatly simplifies the process of finding conforming messages. However, such a choice of the approximation creates the problem that for some input differences, the output difference is always non-zero and there is no effective condition that can make the function behave like the approximation. Mendel et al. managed to overcome this difficulty by using carries in modular additions to compensate for difference bit not following the characteristics. This is not possible when we have only bitwise XORs and it shows an interesting fact that a design that uses XORs instead of modular additions is not always weaker.

4.4 Summary

In this chapter we presented results of our security analysis of SHA-256. This design is significantly more complex than all the previous dedicated hash functions and the results available so far focus only on some selected aspects of the behaviour of the function. It seems however that this is always the first step to any successful analysis and even though we should not expect an attack on the full SHA-256 in the next few years, any advances in our understanding of this design will be noteworthy.

5

Analysis of alternatives: FORK-256

Since source-heavy UFNs with Boolean functions seem to be susceptible to attacks similar to Wang's because only one register is changed after each step and the attacker can manipulate it to a certain extent, one could try designing a hash function using the other flavour of UFNs, namely target-heavy UFNs where changes in one register influence many others. This is the case with designed in 1995 hash function Tiger [3] (tailored for 64-bit platforms) and a recently proposed FORK-256 [81] which is the focus of this chapter.

5.1 A brief description of FORK-256

FORK-256 is a dedicated hash function recently proposed by Hong et al. [81, 82]. It is based on the classical Merkle-Damgård iterative construction with the compression function that maps 256 bits of state and 512 bits of message to 256 bits of a new state. For the complete description we refer interested readers to [81].

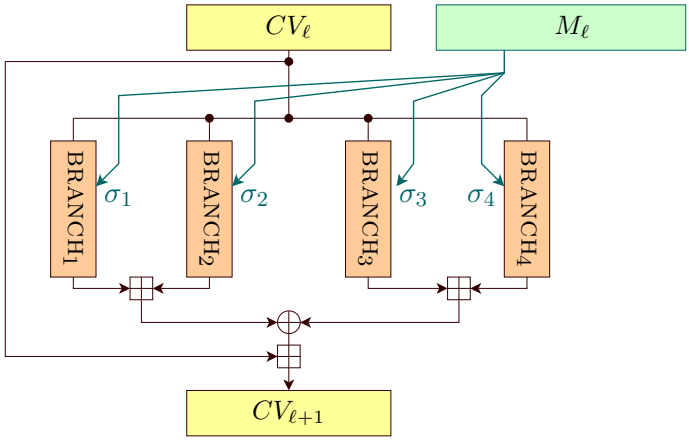


Figure 5.1: Compression function of FORK-256 consists of four parallel branches B1, ..., B4, each one processing the set of message words M_ℓ in different order.

The compression function consists of four parallel branches BRANCH_j, $j = 1, 2, 3, 4$, each one of them using a different permutation σ_j of 16 message words M_i, $i = 0, \ldots, 15$.

The same set of chaining variables $\text{CV} = (A_0, B_0, C_0, D_0, E_0, F_0, G_0, H_0)$ is input in the four branches. After computing outputs of parallel branches

$$h_j = \text{BRANCH}_j(\text{CV}_\ell, M), \quad j = 1, \ldots, 4,$$

the compression function updates the set of chaining variables according to the formula

$$\text{CV}_{\ell+1} := \text{CV}_\ell + [(h_1 + h_2) \oplus (h_3 + h_4)] \quad ,$$

where modular and XOR additions are performed word-wise. This construction is presented in Fig. 5.1 and can be seen as a further extension of the design principle of two parallel lines used in RIPEMD [125].

Each branch function BRANCH_j, $j = 1, 2, 3, 4$ consists of eight steps.

In each step $k = 1, \ldots, 8$ the branch function updates its own copy of eight chaining variables using the step transformation depicted in Fig. 5.2, where $R_i^{(j)}$ denotes the value of the register R in j-th branch after step i. All $A_0^{(j)}, \ldots, H_0^{(j)}$ are initialised with corresponding values of eight chaining variables.

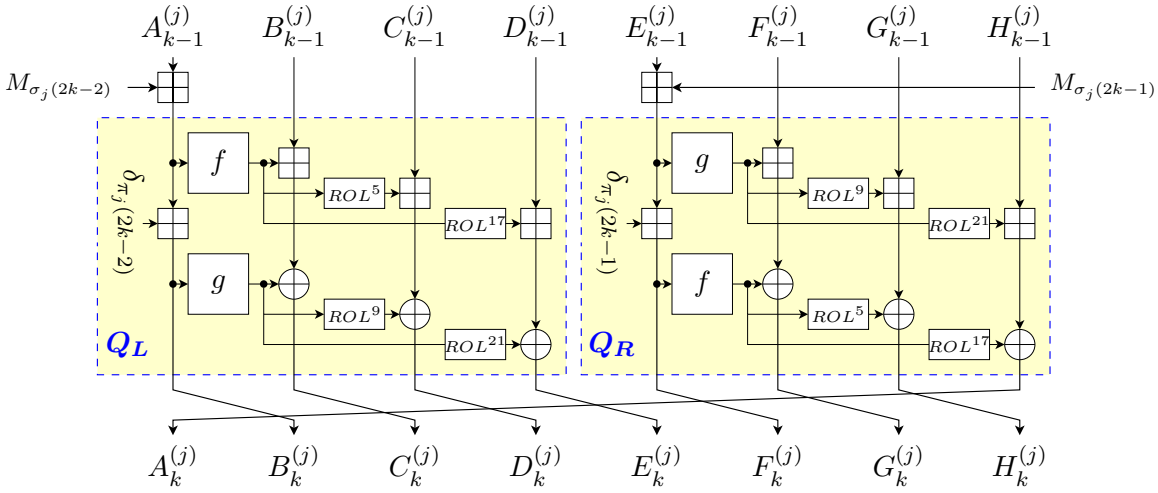


Figure 5.2: Step transformation of a single branch of FORK-256. Q-structures are marked with dashed lines.

Table 5.1: Constants $\delta_0, \ldots, \delta_{15}$ used in FORK-256

δ	0	1	2	3
0	428a2f98	71374491	b5c0fbcf	e9b5dba5
4	3956c25b	59f111f1	923f82a4	ab1c5ed5
8	d807aa98	12835b01	243185be	550c7dc3
12	72be5d74	80deb1fe	9bdc06a7	c19bf174

Functions f and g mapping 32-bit words to 32-bit words are defined as

$$f(x) = x + \left(ROL^7(x) \oplus ROL^{22}(x)\right), \quad g(x) \quad = x \oplus \left(ROL^{13}(x) + ROL^{27}(x)\right) \quad .$$

Constants $\delta_0, \ldots, \delta_{15}$ used in each step are defined as the first 32 bits of fractional parts of binary expansions of cube roots of the first 16 primes and are presented in Table 5.1. Finally, permutations σ_j of message words and permutations π_j of constants are shown in Table 5.2.

Table 5.2: Message and constant permutations used in four branches of FORK-256

j	message permutation σ_j	permutation of constants, π_j
1	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
2	14 15 11 9 8 10 3 4 2 13 0 5 6 7 12 1	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
3	7 6 10 14 13 2 9 12 11 4 15 8 5 0 1 3	1 0 3 2 5 4 7 6 9 8 11 10 13 12 15 14
4	5 12 1 8 15 0 13 11 3 10 9 2 7 14 4 6	14 15 12 13 10 11 8 9 6 7 4 5 2 3 0 1

5.2 Analysis of step transformation of FORK-256

The step transformation described in the previous section can be logically split into three parts: addition of message words, two parallel mixing structures Q_L and Q_R and a final rotation of registers. The key role is played by the two transformations of four words, Q_L and Q_R (marked in Fig. 5.2 with yellow boxes) as they are the main source of diffusion in the compression function. It is clear that if we can find interesting differential characteristics for Q_L and Q_R, we should be able to extend them to the whole branch and maybe also the whole function.

Let us focus on Q_L, as Q_R is very similar to Q_L (f and g are swapped and rotation amounts are different) and the properties we are going to discover work for both of them.

Characteristics of the form $(0, \Delta B, \Delta C, \Delta D) \rightarrow (0, \Delta B, \Delta C, \Delta D)$ are not that difficult to get since in each step differences in registers B, C, D are modified by only one modular addition and one XOR operation. Whether we consider modular or XOR differences, there is only one incompatible operation to deal with.

We can combine such characteristics to get a straightforward differential for up to three steps for each branch.

The difficult part is characteristics of the form $(\Delta A, 0, 0, 0) \rightarrow (\Delta A, 0, 0, 0)$. As far as we could see, there are two ways of finding them. The first method of finding those desired characteristics is based on the fact that both f and g are not bijective so we can hope that we can find such inputs x, x' that $f(x) = f(x')$ and $g(x + \delta) = g(x' + \delta)$. The second one is aimed at getting zero differences in registers B, C, D in spite of non-zero differences at the outputs of f and g. In next sections we describe both of them in detail.

5.3 Simultaneous collisions for f and g

For given value δ, we would like to find all x and x' such that $f(x) = f(x')$ and $g(x + \delta) = g(x' + \delta)$. A naive search would require computations of order 2^{64}, which is well beyond our computing resources. A less naive method trades time

for memory[1]. Below we describe this tradeoff in a way that requires computations of order 2^{32} and 2^{32} memory for the particular functions f and g used in FORK-256. Again, we focus on Q_L, i.e. f is applied before g.

Step 1: We determine which inputs x have more than one preimage. This is done by initialising an array of 2^{32} entries to zero, and then incrementing entries indexed by $f(x)$ in the array for all 2^{32} inputs x. The entries with their values at least 2 are output. There are about 2^{30} of these. In fact, this step is not necessary for our algorithm, but it may be helpful in practice since it reduces memory requirements for the next step.

Step 2: Read in the values of $f(x)$ from Step 1, i.e. the values that have more than one preimage. Then, for each input value, build a linked list of all preimages of that value. This is done in a similar way to Step 1: compute all 2^{32} values of $f(x)$, and for each value that matches one of the inputs from Step 1 (this can be checked quickly with a hash table), add it to the corresponding linked list. The longest linked list for f has 12 preimages.

Step 3: Process the linked lists from Step 2. For each linked list, consider the set of values $x_1, \ldots, x_k$ that map to the same image. See if there is any x_i and x_j in that list such that $g(x_i + \delta) = g(x_j + \delta)$, and if so, output the pair as a solution of simultaneous collisions of f and g.

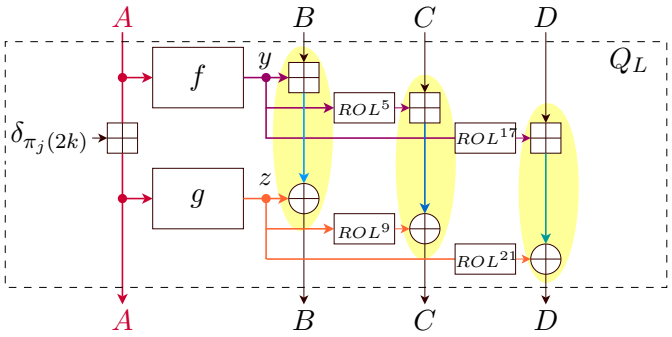
The running time of Step 3 depends upon the number of combinations of pairs of preimages that map to the same image. According to our computations, this is $2134351185 < 2^{31}$.

There are many potential tricks to reduce the search space and/or memory requirements further, but the above algorithm was sufficient to determine the following four solutions for Q_L

$$x = \texttt{4b4d2a05}, \quad x' = \texttt{6ff2f3e9}, \text{ for } \delta_1 = \texttt{71374491},$$
$$x = \texttt{06def69a}, \quad x' = \texttt{aeb691e5}, \text{ for } \delta_2 = \texttt{b5c0fbcf},$$
$$x = \texttt{27a61343}, \quad x' = \texttt{67eac4d8}, \text{ for } \delta_3 = \texttt{e9b5dba5},$$
$$x = \texttt{04549cdc}, \quad x' = \texttt{20d331a5}, \text{ for } \delta_7 = \texttt{ab1c5ed5},$$

[1]The idea of this algorithm is due to Scott Contini

Figure 5.3: A situation where a non-zero output difference of f is cancelled out by a non-zero output difference of g is called a micro-collision. Three simultaneous micro-collisions in Q_L are marked in this picture with yellow clouds.



and two for Q_R

$$x = \texttt{445c5563}, \quad x' = \texttt{d73bc777}, \text{ for } \delta_{10} = \texttt{243185be},$$
$$x = \texttt{be452586}, \quad x' = \texttt{edfd4d5b}, \text{ for } \delta_{14} = \texttt{9bdc06a7}.$$

This is the complete list of solutions and so far it is hard to see how one could use them in any attack. Clearly, we need another method of finding such step differentials.

5.4 Micro-collisions in Q_L and Q_R

In this section we concentrate on an alternative way of finding characteristics of the form $(\Delta A, 0, 0, 0) \rightarrow (\Delta A, 0, 0, 0)$ in Q_L and show that it works for Q_R as well. The idea is to look for pairs of inputs to the register A and appropriate values of registers B, C, D such that output differences in registers B, C, D are equal to zero in spite of non-zero differences at the outputs of functions f and g. Such a situation is possible if we have three simultaneous micro-collisions: differences in g cancel out differences from f in all three registers B, C, D as shown in Fig. 5.3.

5.4.1 Necessary and sufficient condition for micro-collisions

Let us denote $y = f(x)$, $y' = f(x')$ and $z = g(x + \delta)$, $z' = g(x' + \delta)$. We have a micro-collision in the first line if the following equation is satisfied

$$(y + B) \oplus z = (y' + B) \oplus z' \tag{5.1}$$

for given y, y', z, z' and some constant B. Our aim is to find the set of all constants B for which (5.1) is satisfied.

Let us first introduce three different representations of differences between two numbers $x, x' \in \mathbb{Z}_{2^{32}}$. We will use certain relationships between them in our analysis.

- The first kind of representation is the usual XOR difference. We will treat it as a vector of 32 digits representing bits of $x \oplus x'$ and denote it $\Delta^{\oplus}(x, x') \in \{0, 1\}^{32}$.
- The second one is a plain integer difference. For two numbers x, x', we define the integer difference ∂x simply as the result of the subtraction of two operands, i.e. $\partial x = x - x'$, $-2^{32} < \partial x < 2^{32}$.
- The third kind of representation we will be using is the signed binary representation. It uses three digits, 1, 0, -1, and a pair x, x' has signed binary representation $\Delta^{\pm}(x, x') = (x_0 - x'_0, x_1 - x'_1, \ldots, x_{31} - x'_{31})$, i.e. the i-th component is the result of the subtraction of corresponding bits of x and x' at position i.

A simple but important observation is that if a difference has the signed representation $(r_0, r_1, \ldots, r_{31})$ then the corresponding XOR difference has the form $(|r_0|, |r_1|, \ldots, |r_{31}|)$, i.e. the XOR difference has ones in those places where the signed difference has a non-zero digit, either -1 or 1.

The relationship between integer and signed binary representations is more interesting. An integer difference ∂x corresponds to a signed binary representation $(r_0, \ldots, r_{31})$ if $\partial x = \sum_{i=0}^{31} 2^i \cdot r_i$ where $r_i \in \{-1, 0, 1\}$. Of course this correspondence is one-to-many because of the integer difference–preserving transformations of signed representations, $(*, 0, 1, *) \leftrightarrow (*, 1, -1, *)$ and $(*, 0, -1, *) \leftrightarrow$

$(*, -1, 1, *)$, that can stretch or shrink chunks of ones. To see this better consider a small example. Let us assume words of 4 bits and consider $\Delta^{\pm}(11, 2) = (1, 0, 0, 1)$, $\Delta^{\pm}(14, 5) = (1, 0, 1, -1)$ and $\Delta^{\pm}(12, 3) = (1, 1, -1, -1)$. All these binary signed representations correspond to the integer difference $\partial x = 9$. Note that we can go from one pair of values to another by adding an appropriate constant, e.g. $(12, 3) = (11 + 1, 2 + 1)$. This addition preserves the integer difference but can modify the signed binary representation.

After this introductory part we are equipped with the necessary tools and can go back to our initial problem. Rewriting (5.1) as

$$(y + B) \oplus (y' + B) = z \oplus z' \ . \tag{5.2}$$

we can easily see that the signed difference $\Delta^{\pm}(y + B, y' + B)$ can have non-zero digits only in those places where the XOR difference $\Delta^{\oplus}(z, z')$ has ones. This narrows down the set of all possible signed binary representations that can "fit" into XOR difference of a particular form to $2^{h_w(\Delta^{\oplus}(z,z'))}$. But since a single signed binary representation corresponds to a unique integer difference, there are also only $2^{h_w(\Delta^{\oplus}(z,z'))}$ integer differences ∂y that "fit" into the given XOR difference $\Delta^{\oplus}(z, z')$ and what is important, integer differences are preserved when adding a constant B.

Thus, to check whether a particular difference $\partial y = y - y'$ may "fit" into XOR difference we need to solve the following problem: given $\partial y = y - y'$, $-2^{32} < \partial y < 2^{32}$ and a set of positions $I = \{k_0, k_1, \ldots, k_m\} \subset \{0, \ldots, 31\}$ (that is determined by non-zero bits of $\Delta^{\oplus}(z, z')$), decide whether it is possible to find a binary signed representation $r = (r_0, \ldots, r_{31})$ corresponding to ∂y such that

$$\partial y = \sum_{i=0}^{m} 2^{k_i} \cdot r_{k_i} \quad \text{where } r_{k_i} \in \{-1, 1\} \ . \tag{5.3}$$

Substituting $t_i = (r_{k_i} + 1)/2$ we can rewrite the above equation in the equivalent form

$$\partial y + \sum_{i=0}^{m} 2^{k_i} = 2^{k_0+1} t_0 + 2^{k_1+1} t_1 + \cdots + 2^{k_m+1} t_m \ , \tag{5.4}$$

where $t_i \in \{0, 1\}$. Deciding if there are numbers t_i that satisfy (5.4) is an instance

of the knapsack problem and since it is superincreasing (because weights are powers of two), we can do this very efficiently.

This gives us a computationally efficient necessary condition for micro-collision in a line: if $\partial y = y - y'$ cannot be represented as (5.3), no constant B exist and there is no solution of (5.1).

Moreover, we can show that this is as well a sufficient condition: if we can find a solution to the problem (5.3), then there exist a constant B that modifies the signed difference in such a way that it "fits" the prescribed XOR pattern.

Observe that since the solution of the superincreasing knapsack problem (5.4) is unique, so is the solution of the equivalent problem (5.3). This means that we know the unique signed representation $\Delta^{\pm}(u, u + \partial y) = (r_0, \ldots, r_{31})$ that is compatible with the XOR difference $\Delta^{\oplus}(z, z')$ and yields the integer difference ∂y. However, a unique signed representation corresponds to a number of concrete pairs $(u, u + \partial y)$. If at a particular position $j \in I$ we have $r_j = -1$, we know that in this position the value of j-th bit of u has to change from 1 to 0. Similarly, if we have $r_j = 1$, the j-th bit of u should change from 0 to 1. The rest of the bits of u (corresponding to positions with zeros in $\Delta^{\pm}(u, u + \partial y)$) can be arbitrary. That way we can easily determine the set $\mathcal{U}$ of all such values u. It is clear that $\mathcal{U}$ always contains at least one element.

Now, since $u = y + B$ for all $u \in \mathcal{U}$, the set $\mathcal{B}$ of all constants B satisfying (5.1) is simply $\mathcal{B} = \{u - y : u \in \mathcal{U}\}$.

This reasoning shows also that if we can have a micro-collision in a line, there are $|\mathcal{B}| = 2^{32-h_w(z \oplus z')}$ constants that yield the micro-collision if the most significant bit of $z \oplus z'$ is zero and $2^{32-h_w(z \oplus z')+1}$ if the MSB of $z \oplus z'$ is one. The difference is caused by the fact that if $31 \in I$, we do not need to change u_{31} in a particular way (i.e. either $1 \to 0$ or $0 \to 1$), any change is fine since we do not introduce carries anyway.

Finally, since we didn't use any properties of functions f and g, the same line of argument applies not only to micro-collisions in Q_R but also to the same structure with any functions in places of f and g.

Example Let us consider an example of a micro-collision happening in Q_L of the first step in branch 1 shown in Fig. 5.4. For two inputs $x =$ `0x15cce045` and

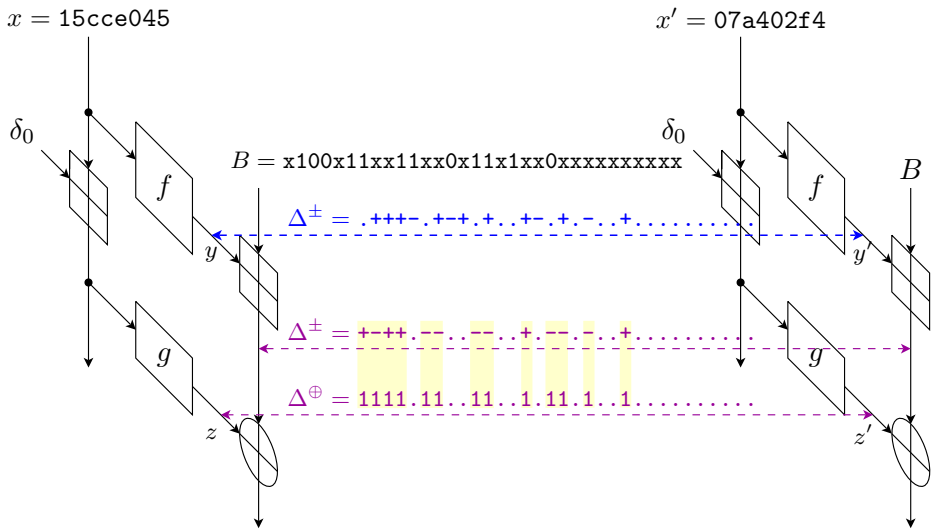


Figure 5.4: An example of a micro-collision. Singed binary representation $\Delta^{\pm}$ of a modular difference $\partial y = y - y'$ is modified by adding a constant B so that it "fits" into XOR difference $\Delta^{\oplus}(z, z')$.

$x' =$ `0x07a402f4` we have $y = f(x) =$ `0x0d0231f7`, $y' = f(x') =$ `0x76a495f7` and the integer difference between them is equal to $\partial y = y' - y = 1772250112$.

The outputs of the function g are $z =$ `0x94e96c55` and $z' =$ `0x628cc855` and the corresponding XOR difference is

$$\Delta^{\oplus}(z, z') = \texttt{11110110011001011010010000000000}\ .$$

We can find a micro-collision iff we can find coefficients $a_i \in \{-1, 1\}$ such that

$$1772250112 = \sum_{i \in I} a_i \cdot 2^i$$

where $I = \{31, 30, 29, 28, 26, 25, 22, 21, 18, 16, 15, 13, 10\}$ is the set of positions with non-zero digits of $\Delta^{\oplus}(z, z')$. In our case it is possible since

$$1772250112 = 2^{31} - 2^{30} + 2^{29} + 2^{28} - 2^{26} - 2^{25} - 2^{22} - 2^{21} + 2^{18} - 2^{16} - 2^{15} - 2^{13} + 2^{10}.$$

It can be written more compactly as a signed binary difference

$$\Delta^{\pm}(y + B, y' + B) = \texttt{+-++.--..--..+.--.-..+.....}$$

where sign '+' stands for $+2^i$ and '−' for -2^i. Now we know that a micro-collision is possible and want to find the set of "good" constants B that make it happen. Looking at the signed difference above, we can see that if the bit difference is '+' i.e. $0 \to 1$ than the corresponding bit of $y + B$ has to be 0, for '−' we need transition $1 \to 0$ and the bit has to be equal 0. This means that we can represent the set of all possible values $y + B$ as

$$y + B = \texttt{x100x11xx11xx0x11x1xx0xxxxxxxxxx}$$

where 'x' denotes any digit, zero or one. From this we can easily get B by subtracting y from one of the possible choices for $y + B$.

5.4.2 Estimation of probabilities of micro-collisions

From a practical point of view, we are interested in the probability that a random pair of values (A, A') may lead to simultaneous micro-collisions and what is the overall probability of characteristics of the form $(\Delta A, 0, 0, 0) \to (\Delta A, 0, 0, 0)$ when we cannot manipulate the values of registers A, B, C, D.

We conducted some experiments for Q_L and Q_R with different constants δ. Our results indicate that the probability that a random pair of inputs (A, A') may lead to simultaneous micro-collisions in all three lines is around 2^{-23} with probability for a single line close to 2^{-13}.

The probability that random constants B, C, D adjust the difference in $f(x)$ properly depends on Hamming weights of $\Delta^{\oplus}(z, z')$. One example of such distribution of weights obtained by testing 2^{32} random pairs[2] is presented in Table 5.3. We can see a clear peak around weights 24–26, so, according to the formula describing the size of the set of constants from the previous subsection, we can expect $2^6 \sim 2^8$ "good" constants in each of the sets $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$ and thus the probability that a random constant falls into that set is around $2^{-24} \sim 2^{-26}$. Of course to get a result for all three branches we need to cube that number.

Using the above results, we can try to estimate the probability that a set of three simultaneous micro-collisions occurs if we have no control of any values

[2]In all experiments we were using Mersenne Twister [102] as the source of pseudorandom numbers

Table 5.3: Distribution of Hamming weights of $\Delta^{\oplus}(g(x+\delta_0), g(x'+\delta_0))$ corresponding to potential simultaneous micro-collisions after testing 2^{32} random pairs x, x'.

h_w	0	1	...	18	19	20	21	22	23
count	1	0	...	0	1	6	18	29	59
h_w	24	25	26	27	28	29	30	31	32
count	78	74	90	56	39	14	1	0	0

A, A', B, C, D. Multiplying 2^{-23} by $2^{-72} \sim 2^{-78}$ we get an estimation of $2^{-95} \sim 2^{-101}$. It shows that such differentials are not immediately useful, but if we can force specific values of registers to desired values, they may be used to construct collisions for at least simplified variants of FORK, as presented in next sections.

5.5 Finding high-level differential paths in FORK-256

In this section we present a general strategy of finding high-level differentials in the structure of FORK-256 provided that we can prevent the differences from spreading from A and E to other registers. We start with a basic intuition and later on formalise it into the model using simple linear algebra and coding theory tools. Later on, we show that we can generalise this model to improve the probability of the paths by relaxing some of the initial conditions.

5.5.1 Basic intuition

If we can avoid mixing introduced by the structures Q_L and Q_R (i.e. we know how to get differentials $(\Delta A, 0, 0, 0) \rightarrow (\Delta A, 0, 0, 0)$ and $(\Delta E, 0, 0, 0) \rightarrow (\Delta E, 0, 0, 0)$) and we can assume that differences in the registers B, C, D and F, G, H remain unchanged, the only places where differences can change are registers A and E, after the addition of a message word difference. Thus, the values of registers in steps are simple linear functions of registers of the initial vector and message words. If we denote $\Delta X_0 + \Delta M_{\sigma_j(a)}$ by [X,a] and $\Delta X_0 + \Delta M_{\sigma_j(a)} + \Delta M_{\sigma_j(b)}$ by [X,a,b], where σ_j is the permutation of message words used in branch $j = 1, 2, 3, 4$,

Table 5.4: If no mixing through Q_L and Q_R occurs, differences in registers are combinations of differences in initial vectors and message words. [X,i] stands for $\Delta X_0 + \Delta M_{\sigma_j(i)}$ and [X,a,b] stands for $\Delta X_0 + \Delta M_{\sigma_j(a)} + \Delta M_{\sigma_j(b)}$

	registers							
step	ΔA	ΔB	ΔC	ΔD	ΔE	ΔF	ΔG	ΔH
1	[A,0]	[B]	[C]	[D]	[E,1]	[F]	[G]	[H]
2	[H,2]	[A,0]	[B]	[C]	[D,3]	[E,1]	[F]	[G]
3	[G,4]	[H,2]	[A,0]	[B]	[C,5]	[D,3]	[E,1]	[F]
4	[F,6]	[G,4]	[H,2]	[A,0]	[B,7]	[C,5]	[D,3]	[E,1]
5	[E,1,8]	[F,6]	[G,4]	[H,2]	[A,0,9]	[B,7]	[C,5]	[D,3]
6	[D,3,10]	[E,1,8]	[F,6]	[G,4]	[H,2,11]	[A,0,9]	[B,7]	[C,5]
7	[C,5,12]	[D,3,10]	[E,1,8]	[F,6]	[G,4,13]	[H,2,11]	[A,0,9]	[B,7]
8	[B,7,14]	[C,5,12]	[D,3,10]	[E,1,8]	[F,6,15]	[G,4,13]	[H,2,11]	[A,0,9]
out	[A,0,9]	[B,7,14]	[C,5,12]	[D,3,10]	[E,1,8]	[F,6,15]	[G,4,13]	[H,2,11]

we can write this down concisely in a tabular form presented in Table 5.4.

It is clear that differences in registers at any particular step are combinations of differences introduced in the initial vector $(A_0, \ldots, H_0)$ and differences in message words $M_0, \ldots, M_{15}$.

5.5.2 Constructing the model

If we consider the simplest case and assume (very optimistically) that any two differences can cancel each other (this is the case with XOR differences), we are in fact working over $\mathbb{F}_2$ and differences in all registers are $\mathbb{F}_2$-linear combinations of differences $\Delta A_0, \ldots, \Delta H_0$ and $\Delta M_0, \ldots, \Delta M_{15}$ (which are now seen as elements of $\mathbb{F}_2$). Now output differences of the whole compression function (including feed-forward) are also linear combinations of differences from $S = (\Delta A_0 \ldots, \Delta H_0, \Delta M_0, \ldots, \Delta M_{15})$ and we can represent this map as an $\mathbb{F}_2$-linear function, $(\Delta A, \ldots, \Delta H) = L_{out}(S)$. This means we can easily find the set $\mathcal{S}_c$ of all vectors $S = (\Delta A_0 \ldots, \Delta H_0, \Delta M_0, \ldots, \Delta M_{15})$ that yield zero output differences at the end of the function simply as the kernel of this map, $\mathcal{S}_c = \ker(L_{out})$.

To minimise the complexity of the attack, we want to find high-level paths as short as possible. Since each register difference in each step is a linear function of differences $\Delta A_0 \ldots, \Delta H_0, \Delta M_0, \ldots, \Delta M_{15}$ and there are only 2^{24} of them, the straightforward approach is to enumerate them all and for any desirable subset of registers (e.g. for collisions in two or three branches) count the number of registers containing non-zero differences and pick those differences S that give

the smallest one. This straightforward process can be improved. If we denote by V the vector of register states we are interested in, there is a matrix Ψ such that $V = S \cdot \Psi$. The matrix Ψ can be seen as a generator matrix of a linear code over $\mathbb{F}_2$. Minimum words of that code correspond to register states with minimal weight. To find collisions (or other restricted paths), the appropriate generating matrix is $Basis(\ker L_{out}) \cdot \Psi$ (or $Basis(\ker(L)) \cdot \Psi$ where L is the linear map describing those registers we want to be zero). Here $Basis(A)$ denotes the basis matrix of a linear space A. Using systems like MAGMA [26], finding minimum words in such codes takes only a fraction of a second.

Our computations show that

- The minimal *collision* path in branches 1-2 uses differences in M_0 and M_9,
- The minimal *collision* path in branches 3-4 uses differences in M_{14} and M_{15},
- The minimal *collision* path for all four branches requires differences in message words M_6 and M_{12},
- The minimal *unrestriced* path for all branches has differences in the message M_{12} only.

However, differences in registers other than A and E don't contribute to the complexity of the attack that much. The measure based on the number of differences in registers A and E only corresponds more closely to the number of "difficult" differentials we need to handle that require finding micro-collisions. Considering this, we also conducted experiments for different variants of FORK-256 counting only differences in registers A and E.

The results are presented in Table 5.5. The first column specifies whether we are interested in collision, pseudo-collisions (differences also appear in the initial vector) of just a free path – no specific conditions on differences are imposed. The third column gives the minimal number of Q–structures that require special differentials and thus also micro-collisions in registers B,C,D or F,G,H. The last column gives an example of message and/or chaining variables differences that induce the high-level path with the given number of sets of micro-collisions.

Table 5.5: Minimal numbers m of sets of simultaneous micro-collisions in Q_L and Q_R necessary in different attack scenarios on variants of FORK-256

Scenario	Branches	m	Differences in
Collisions	1,2	2	M_0, M_9
Collisions	3,4	2	M_{14}, M_{15}
Collisions	1,3	3	M_5
Collisions	1,4	3	M_2
Collisions	2,3	3	M_3
Collisions	2,4	3	M_9
Pseudo-collisions	1,2,3	6	B_0
Pseudo-collisions	1,2,4	6	B_0
Pseudo-collisions	1,3,4	6	B_0
Pseudo-collisions	2,3,4	6	B_0
Collisions	1,2,3,4	12	M_6, M_{12}
Free path	1,2,3,4	6	M_{12}

5.5.3 More general variant of path finding

We can generalise this approach even further. Depending on whether we force a micro-collision to happen in a particular line or not, we have eight different models for each Q-structure. Using the linear model that assumes that all differences cancel each other, we can express output differences of each Q_L-structure as

$$\begin{aligned}
\Delta A_{i+1} &= \Delta A_i \\
\Delta B_{i+1} &= \Delta B_i + q_B \cdot \Delta A_i \\
\Delta C_{i+1} &= \Delta C_i + q_C \cdot \Delta A_i \\
\Delta D_{i+1} &= \Delta D_i + q_D \cdot \Delta A_i
\end{aligned}$$

where $q_B, q_C, q_D \in \mathbb{F}_2$ are fixed coefficients characterising the Q_L-structure. The same is true for Q_R-structures. This means that we have 8^{64} possible linear models of FORK-256 when we allow such varied micro-collisions to happen. In fact, results presented in Table 5.5 correspond to a special case when all coefficients q are equal to zero. Relaxing this condition and allowing for micro-collisions in only selected lines decreases the number of active Q-structures, however, at the expense of additional conditions required to cancel differences coming from different parts of the structure.

Table 5.6: Minimal numbers m of Q-structures with micro-collisions for different scenarios of finding generalised high-level differential paths. Q-structures are numbered from 1 to 64 where 1 corresponds to Q_L in the first step of branch 1 and 64 to Q_R in the last step od branch 4. P-C in scenario means finding pseudo-collisions and C collisions.

Scenario	Branches	m	Differences in	active Q-structures
P-C	1,2,3,4	5	$IV[7], M_2, M_{11}$	12:000, 25:000, 35:001, 41:001, 51:010
C	1,2,3,4	6	M_{12}	13:000, 31:001, 40:000, 47:100, 50:000, 57:000
P-C	1,2,3	2	$IV[1], M_{12}$	8:100, 24:0
	1,2,4	3	$IV[7], M_{11}$	3:000, 51:010, 60:000
	1,3,4	3	$IV[7], M_2$	35:001, 44:000, 51:000
	2,3,4	3	$IV[3], M_9$	36:010, 43:000, 52:000
C	1,2,3	3	M_0, M_3, M_9	1:001, 20:010, 39:100
	1,2,4	4	M_1, M_2	2:001, 9:000, 25:100, 51:000
	1,3,4	5	M_9	10:000, 39:001, 42:001 43:010, 59:000
	2,3,4	5	M_3, M_9	20:010, 27:000, 39:000 57:000, 59:010

Results of our search for such paths are summarised in Table 5.6. They show that by introducing such an extended model of Q-structures we can significantly decrease the number of necessary micro-collisions. Particularly interesting is the result showing that, under favourable conditions, collisions can be achieved by using a single difference in M_{12} with 6 micro-collision places in the path. In section 5.7 we show how to use this situation to generate near-collisions and theoretically also collisions.

5.6 Collisions for two branches of FORK

We can use the minimal path for branches 1 and 2 to get collisions for these two branches of FORK-256. The idea is to find two related simultaneous micro-collisions, the first one of type f - δ_0 - g (f is followed by δ_0 and then by g) to be used in the left part of the first step of branch 1 and the other one of type g - δ_{12} - f to be used in step 2 of branch 2.

If we can find a pair of values (x, x') that yields f - δ_0 - g micro-collisions and a pair (y, y') that yields g - δ_{12} - f micro-collisions such that the values satisfy the

condition $x - x' = y' - y$, we can construct a collision for branches 1 and 2 by preserving differences $\partial x = x - x'$ in steps 2, 3, 4 of branch 1 and $\partial y = y - y'$ in steps 3, 4, 5 of branch 2. This view is presented in Fig. 5.5.

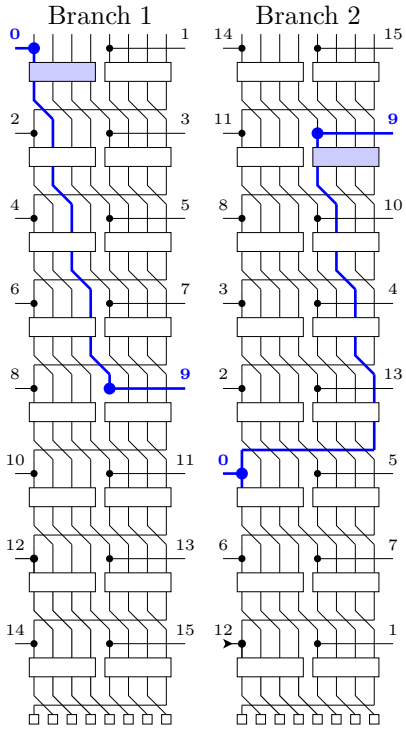


Figure 5.5: Collisions for two branches of FORK-256 can be obtained by introducing modular differences in M_0 and M_9 and finding two sets of micro-collisions.

The algorithm works as follows:

1. find a pair of values x, x' that produce f - δ_0 - g simultaneous micro-collisions and determine the three compatible constants ρ_1, ρ_2, ρ_3, (this step requires around 2^{23} tests of random pairs x, x')

2. for the fixed difference $\partial x = x - x'$ test pairs of the form $y, y' = y + \partial x$ until a simultaneous micro-collision of type g - δ_{12} - f is found. Determine compatible constants τ_1, τ_2, τ_3. (Again, experiments suggest that the complexity of this step is 2^{23} tests)

3. set $IV[1] := \rho_1$, $IV[2] := \rho_2$, $IV[3] := \rho_3$,

4. compute $M_0 := x - IV[0]$, $M'_0 := x' - IV[0]$,

5. set both M_{15} and M'_{15} to $\tau_1 - IV[4] - \delta_{14}$,

6. compute initial values $IV[5]$ and $IV[6]$ as follows

$$\begin{aligned} IV[5] &:= (\tau_2 \oplus f(IV[4] + M_{15} + \delta_{14})) - g(IV[4] + M_{15}), \\ IV[6] &:= (\tau_3 \oplus ROL^5(f(IV[4] + M_{15} + \delta_{14}))) - ROL^9(g(IV[4] + M_{15})) \end{aligned}$$

7. compute the values $M_9 := y - E_1^{(2)}$ and $M'_9 := y' - E_1^{(2)}$, where

$$E_1^{(2)} = ((IV[3] + ROL^{17}(f(IV[0] + M_{14}))) \oplus ROL^{21}(g(IV[0] + M_{14} + \delta_{15}))),$$

   is the value of register E after step 1 in branch 2.

8. preserve the difference ∂x by forcing the value of g to zero in steps 2, 3, 4 (XOR-ing with zero doesn't change the modular difference)

   - set $M'_2 := M_2 := -A_1^{(1)} - \delta 2$,
   - set $M'_4 := M_4 := -A_2^{(1)} - \delta 4$,
   - set $M'_6 := M_6 := -A_3^{(1)} - \delta 6$,

9. similarly, preserve the difference ∂y by forcing the value of f to zero in steps 3, 4, 5 of branch 2

   - set $M'_{10} := M_{10} := -E_2^{(2)} - \delta_{10}$,
     - $\diamond$ in step 3 we cannot modify the value of M_4 as it is already fixed by correction done in branch 1. However, we can modify freely the value of M_8 (and M'_8) which indirectly influences the value of $E_3^{(2)}$ we need to adjust. We do this until the difference in $H_4^{(2)}$ is equal to the difference at the beginning of the step, i.e. in $G_3^{(2)}$. If we exhaust all possible values of M_8, we can modify the value of M_{11} and go to step 9 or pick another constant ρ_1 and start over from step 3.
   - set $M'_{13} := M_{13} := -E_4^{(2)} - \delta_6$,

The complexity of the attack on branches 1 and 2 depends on the effort to find suitable pair of micro-collisions and the amount of work necessary to find the appropriate value of M_8 in step 9.$\diamond$. Microcollisions can be precomputed using around 2^{23} evaluations of functions f, g. The only part we need to deal with

Table 5.7: Example of a pair of messages producing collisions for the first two branches of FORK-256.

IV	6a09e667 ff03f03a f7da19f9 a19f937d 510e527f d1075199 c4bba02c 00000000
M	97770819 00000000 90e31bf1 00000000 e9b1a3b9 00000000 36ca5a85 00000000
	000024a1 6ff47b82 3f7bfaf6 00000000 00000000 014b4e3b 00000000 980100ed
M'	b479fad2 00000000 90e31bf1 00000000 e9b1a3b9 00000000 36ca5a85 00000000
	000024a1 52f188c9 3f7bfaf6 00000000 00000000 014b4e3b 00000000 980100ed

during the attack is the step 9.$\diamond$. In our experiment we had to test ≈ 10000 values of M_8 to find the right one. Since one test is roughly equivalent to computing single step in one branch of FORK (1/32 of the whole function), we can estimate the complexity of 9.$\diamond$ to be less than 2^9 evaluations of the compression function.

This algorithm (partially) uses the following variables: $IV[1]$, $IV[2]$, $IV[3]$, $IV[5]$, $IV[6]$, M_0, M_2, M_4, M_6, M_8, M_9, M_{10}, M_{13}, M_{15}. The following variables can have arbitrary values: $IV[0]$, $IV[4]$, $IV[7]$, M_1, M_3, M_5, M_7, M_{11}, M_{12}, M_{14}.

Finally, we present an example of a collision in Table 5.7.

Collisions for branches 3 and 4 can be obtained using exactly the same method by introducing appropriate differences in message words M_{14} and M_{15}.

A similar approach was used independently by Mendel et al. [107] who also discovered the existence of pathological differentials and made use of them to produce collisions for two branches of FORK-256.

5.7 Collisions for the full compression function

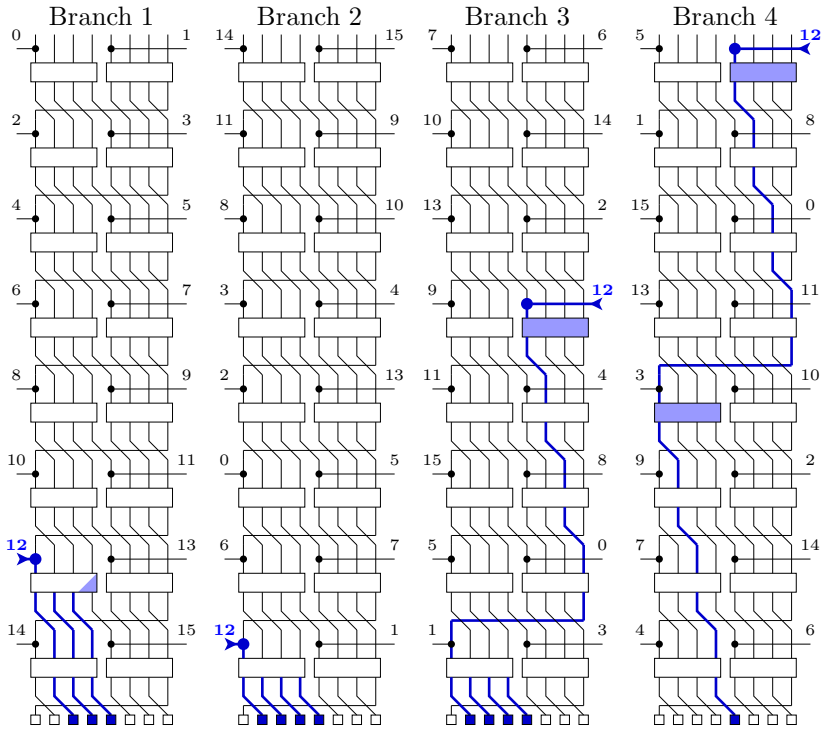
In this section we show how a high-level path using differences in M_{12} presented in section 5.5 can be used to find very low weight output differences of the compression function of FORK-256 and we argue that this approach might be applicable to finding full collisions faster than using the birthday paradox.

5.7.1 Overview

The foundation of our attack is the observation that if we introduce a difference in M_{12} and we are able to prevent it from propagating to other registers in step 1 and step 5 of branch 4 and in step 4 of branch 3 and it does not introduce a difference in register E_7 of branch 1 then the output difference is confined to

registers B,C,D and E only, ie. to at most 128 bits in total. This situation is illustrated in Figure 5.6.

Figure 5.6: High-level path used to find near-collisions in FORK-256. Thick lines show the propagation of differences. Q-structures for which micro-collisions have to be found are greyed out.



We can decrease the number of affected bits further by selecting a modular difference having differences in only a few most significant bits as the difference in register B will be restricted to only those most significant bits as well.

Now, if we can find pairs of messages satisfying aforementioned conditions efficiently enough and we can assume that output differences have distribution close to uniform, we can expect to find very low weight differences and ultimately also a collision.

The attack consist of two phases. During the first one, we find simultaneous micro-collisions in steps 1 and 5 of branch 4 and step 4 of branch 3 for a specially selected modular difference introduced in M_{12}.

In the second phase we use free message words M_4 and M_9 that do not interfere with already fixed micro-collisions in branches 3 and 4 to find messages

that yield no difference in register $E_7^{(1)}$ due to a single micro-collision in line D in step 7 of branch 1.

5.7.2 Achieving micro-collisions in branches 3 and 4

We assume that we have already chosen a suitable modular difference d. We proceed as follows.

- We start with branch 4. We find x_1 such that $x_1, x_1 + d$ give simultaneous g - δ_{15} - f micro-collisions for step 1 of branch 4, compute corresponding constants τ_1, τ_2, τ_3 and assign $IV[5] := \tau_1$, $IV[6] := \tau_2$, $IV[7] := \tau_3$. Set M_{12} to $x_1 - IV[4]$ and M'_{12} to $x_1 - IV[4] + d$.

- Fix arbitrary values of M_5, M_1, M_8, M_{15}, M_0, M_{13} and M_{11} and compute the first half of the branch, up to step 5. Then, in step 5 find a pair of values $x_2, x_2 + d^*$ (where $d^* = A_4^{(4)} - A_4'^{(4)}$ is the modular difference in register A after step 4) yielding simultaneous f - δ_6 - g micro-collisions and compute corresponding constants ρ_1, ρ_2, ρ_3. If no such solution exists, repeat this point, otherwise, set $M_3 := x_2 - A_4^{(4)}$.

- By manipulating message words M_1, M_{15}, M_{13} (and also M_0 and M_{11}) we need to adjust the values of registers $B_4^{(4)}$, $C_4^{(4)}$, $D_4^{(4)}$ to ρ_1, ρ_2, ρ_3.

  - Since $B_4^{(4)} = A_3^{(4)} + M_{13} + \delta_8$ and we want $B_4^{(4)} = \rho_1$, we adjust the value of $B_4^{(4)}$ by setting $M_{13} := \rho_1 - A_3^{(4)} - \delta_8$.

  - Now, starting from ρ_2 we can go back one step and compute the necessary value of $\bar{B}_3^{(4)} = [\rho_2 \oplus g(A_3^{(4)} + M_{13} + \delta_8)] - f(A_3^{(4)} + M_{13})$. We can do this by setting $M_{15} := \bar{B}_3^{(4)} - A_2^{(4)} - \delta_{10}$. This change has also influenced the value of $E_3^{(4)}$ so we have to compensate it by adjusting the value of M_{11}.

  - Similarly, going back from ρ_3 two steps we can determine the necessary value of $B_2^{(4)}$ and adjust M_1 accordingly. Again, we need to compensate the induced change in $E_2^{(4)}$ by adjusting the value of M_0 and the change in $E_3^{(4)}$ by correcting M_{11} again.

- Now we switch to branch 3. We choose values $x_3, x_3 + d$ that cause simultaneous g - δ_6 - f micro-collisions in step 4 and find corresponding constants $\lambda_1, \lambda_2, \lambda_3$. Using them we compute the necessary values of registers E–H, ie. $\bar{E}_3^{(3)} := x_3 - M_{12}$, $\bar{F}_3^{(3)} := \lambda_1$, $\bar{G}_3^{(3)} := \lambda_2$, $\bar{H}_3^{(3)} := \lambda_3$.

- Again, by going backwards and adjusting message words M_2, then M_{14} and then M_6 and M_{10} and finally $IV[1]$ in a similar manner to what we did in branch 4 we obtain desired values of register at the beginning of step 4.

- Since we have just modified $IV[1]$ we need to go back to branch 4 and compensate for this change by adjusting the value M_{11} once again.[3]

After this procedure we have obtained a differential path in branches 3 and 4 presented in Figure 5.6. The important fact is that changing the values of message words M_4 and M_9 does not change this path, so after fixing branches 3 and 4 we still have 64 bits of freedom left. Also note that there are many possible states of branches 3 and 4 following this path as at the beginning of the process we can select many arbitrary values, e.g. M_5, M_8, M_7 and $IV[0], IV[2], IV[4], IV[5]$. Additionally, there are many constants to choose from when fixing a micro-collision.

5.7.3 Single micro-collision in branch 1

What is left are branches 1 and 2. Fortunately, we do not need to pay attention to branch 2 at all as M_{12} appears in the very last step and so in any case it induces differences in registers B–E only.

In branch 1 we need a single micro-collision in the third line of step 7. It seems to us that there is no better way of finding messages that cause that micro-collision to happen than by randomly testing message words M_4 and M_9.

The probability of the success heavily depends on the modular difference in use. A few best modular differences we could find are presented in Table 5.8.

Let us analyse the computational complexity of finding this single micro-collision in terms of numbers of full FORK-256 evaluations. Denote by η the number of allowable values for the modular difference in use. By an allowable

[3] Note that this description mentions modification of M_{11} three times. Only the last one is necessary, but we include them all to make the process more intelligible by clear invariants.

Table 5.8: Best modular differences d we could find and their probabilities of inducing a single micro-collision in strand D of step 7 in branch 1. η is the number of input values to strand A that may result in the micro-collision.

difference d	η	observed probability
`0xdd080000`	$2^{21.7}$	$2^{-24.6}$
`0x22f80000`	$2^{21.7}$	$2^{-24.6}$

value we mean an input x for which there exist constants that cause a micro-collision to happen for the pair $(x, x + d)$. For $d =$`22f80000` we have $\eta = 2^{21.7}$ (cf. Table 5.8). We proceed as follows.

- First, we fix the value of M_4. We will exhaust all values of M_9 for this value of M_4.

- Next, step through the computation up until step 7. We need to know all inputs into step 7.

- Then, for each allowable value into strand A of step 7 (note that M_{12} and M'_{12} are already fixed) we step backwards one step to determine the corresponding "allowable outputs" to strand G in step 5 and we store them in a hash table. For each allowable value we need to compute one XOR, one subtraction and store the element in a hash table, so the work effort for this step is about $1/64 \cdot \eta$ of full FORK-256 evaluations. For $\eta = 2^{21.7}$ the complexity of this step is about $2^{15.7}$.

- We loop through all 2^{32} values of M_9. For each one, we compute the output of G only in step 5 and check if it matches something in the lookup table. If so, we proceed forward to see if it causes the difference to disappear in step 7. If not, we go to the next value of M_9.

  The cost of testing one value of M_9 is less than 1/64 of a full FORK evaluation. Essentially, we are computing less than a single Q-structure (FORK-256 consists of 64 of them). We assumed here that the cost of the table lookup is not exceeding the cost of computing the other parts of the Q-structure that we are omitting (strands F and H), which seems to be a fairly safe assumption. For $\eta < 2^{22}$ values that match the allowable out-

puts we do a little bit more (about one more Q-structure in the left part of step 6), but the dominant term is $2^{32} \cdot 1/64 = 2^{26}$.

- After exhausting the list, we proceed with the next value of M_4.

From the above analysis it follows that we process 2^{32} values of M_9 for the work effort of about 2^{26} full FORK evaluations. Since the observed probability of finding a solution is about $2^{-24.6}$ (cf. Table 5.8), we are getting about $2^{7.4}$ solutions for 2^{26} effort. This is equivalent to about $2^{18.6}$ FORK evaluations per solution.

5.7.4 Finding near-collisions: experimental results

If output differences are distributed uniformly on the positions where they can appear (i.e. part of the register B and registers C, D, E) than we can expect a binomial distribution of their Hamming weights. After generating enough pairs we should be able to find some with exceptionally small weights, which can be called near-collisions.

We implemented this algorithm and performed some searches for such output differences with low weights. Comparison of the distribution of Hamming weights of differences obtained by the means of an experiment with theoretical binomial distribution is presented in Figure 5.7. It seems that the experimental distribution is indeed close to the theoretical one. However, one can see a slight bias towards lower weights.

This phenomena can be explained by looking at Table 5.9 which contains counts of non-zero differences appearing at all bit positions of registers B, C, D and E. The expected number of non-zero differences for a truly random bit stream would be $211867/2 \approx 105933$. All bits of registers C, D and E are very close to that value, but in some bits of register B non-zero differences appear with a little lower probability. This most likely accounts for the bias towards smaller weights.

The best result we obtained so far has the output difference of weight 28 and is presented in Table 5.10. It took about a day of work of an average workstation PC to find it.

Figure 5.7: Distribution of Hamming weights of 211867 output differences generated by the algorithm for $d =$`0x22f80000`.

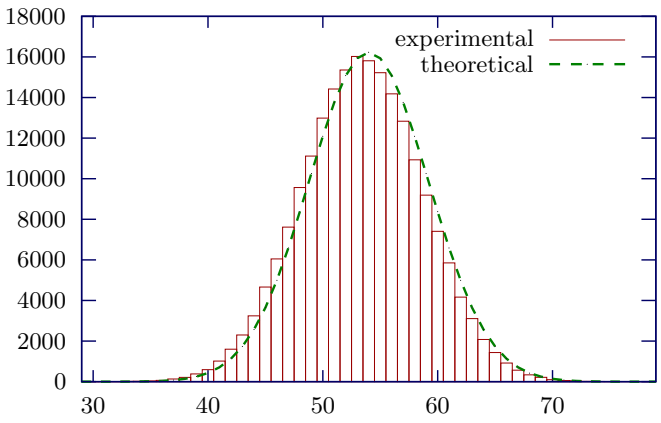


Table 5.9: Counts of non-zero bit differences in registers B, C, D and E after testing 211867 pairs of "close" hashes.

	bits	counts							
B	0 – 7	0	0	0	0	0	0	0	0
	8 – 15	0	0	0	0	0	0	0	0
	16 – 23	0	0	0	0	105812	102276	94549	82277
	24 – 31	68364	105987	102715	105458	97349	89530	105799	103214
C	0 – 7	105460	105722	105723	105751	104016	106018	105458	105833
	8 – 15	105840	104074	106047	105842	105798	104450	99927	106058
	16 – 23	106255	106211	105918	108125	105860	105751	105785	105357
	24 – 31	110069	106080	105834	105726	106008	106559	106134	105892
D	0 – 7	106072	105667	105443	105786	106165	106053	106019	105874
	8 – 15	105949	106556	105629	105597	105709	105308	102826	105302
	16 – 23	105637	105938	105993	104343	105727	106117	105800	105642
	24 – 31	106491	105858	105933	105595	104871	105884	106314	105622
E	0 – 7	105496	105903	105954	105681	106193	105745	105652	105878
	8 – 15	103071	105674	106294	105795	105778	105893	105728	105701
	16 – 23	105913	105857	105977	105725	105963	106232	106061	105743
	24 – 31	106115	105974	107089	105738	105904	106000	105941	105671

5.7.5 Complexity of finding full collisions

Results from the previous subsection suggest that we can consider output differences to be very close to the uniform distribution. Using $d =$`0xdd080000`, there are 109 bits that may contain differences, but we know that differences in bit 19 of register B will always cancel out each other. This means that since at most 108 bits are affected, after generating 2^{108} such pairs we expect to find a collision. We have already computed that the complexity of finding a single pair like this is about $2^{18.6}$ (or less, if better modular differences exist). So the total

Table 5.10: An example of an IV value and a message pair giving a pair of FORK-256 hashes differing on 28 bits.

IV	6a09e667 db1bb914 3c6ef372 a54ff53a 510e527f 767b0824 66410f7d 90f7ce64
M	85a83e55 91d3ca9d a6c2facb 027afd32 000000cb 00000000 9d4a6aba 00000000
	e649c148 4606ae35 6efb18d8 2d6ade8f 1dcb6936 ec995db1 d2ad257b 730f5bb4
M'	85a83e55 91d3ca9d a6c2facb 027afd32 000000cb 00000000 9d4a6aba 00000000
	e649c148 4606ae35 6efb18d8 2d6ade8f 40c36936 ec995db1 d2ad257b 730f5bb4
diff	00000000 8c300000 1d010204 52520104 c0908122 00000000 00000000 00000000

complexity of generating enough pairs to find a collision with high probability is $2^{108} \cdot 2^{18.6} = 2^{126.6}$, more than a factor of two better than the generic birthday attack.

It is worth mentioning that the additional advantage of our attack is that it does not need a huge storage, it requires only about $2 \cdot 2^{22}$ 32-bit words of memory for storing precomputed inputs for micro-collisions and a hash table of similar size.

The above estimate is rather conservative, because if we multiply probabilities of single bit differences being zero (which can be easily derived from Table 5.9) we get the value of $2^{106.4}$ rather than 2^{108} and thus also a lower complexity of the attack of 2^{125} but one has to be cautious as there is no guarantee that the bits are uncorrelated enough to make the computation of this product valid.

5.8 Collisions for the complete hash function

The obstacle for extending the algorithm from Section 5.7 to the full hash function is the requirement for particular values of four chaining values, B_0 required by branch 3 and F_0, G_0, H_0 required by branch 4. Nothing can be done about constants necessary to achieve micro-collision in the first step of branch 4. However, in this section we show that by careful modification of some steps of the procedure we can eliminate the need for choosing the value of the constant B_0. This section is based on the submission [37].

5.8.1 The algorithm

Instead of solving for branch 4, then branch 3, and later making a small adjustment to branch 4 again, the idea is to go through the first step of branch 4 only, then switch to branch 3, and finally return to solve for the rest of branch 4.

Let d denote the modular difference used in M_{12}. Recall that an allowable value x is a value fed to register A (or E) such that there exist constants B, C, D (or F, G, H) that cause simultaneous micro-collisions to happen in all three lines when $x, x + d$ are the values of register A (or E). The modified algorithm first precomputes for difference d all allowable values for step 5 of the left Q-structure of branch 4. Then, the steps are as follows:

Branch 4, step 1 We find x_1 such that $x_1, x_1 + d$ give simultaneous g - δ_{15} - f micro-collisions for step 1 of branch 4, compute corresponding constants τ_1, τ_2, τ_3 and assign $F_0 := \tau_1$, $G_0 := \tau_2$, $H_0 := \tau_3$. Set M_{12} to $x_1 - E_0$ and M'_{12} to $x_1 - E_0 + d$.

Branch 3 We choose values of M_7, M_6, M_{10}, M_{14}, M_{13} and M_2 appearing in the first three steps of branch 3 randomly and compute the function up to the beginning of step 4. We check if the value $E_4^{(3)} + M_{12}$ is an allowable value for the g - δ_6 - f micro-collision in step 4, i.e. we test if there exist constants μ_0, μ_1, μ_2 such that the pair $E_4^{(3)} + M_{12}$, $E_4^{(3)} + M_{12} + d$ yields micro-collisions when those constants are set in registers $F_3^{(3)}$, $G_3^{(3)}$, $H_3^{(3)}$. If it is not, we pick fresh values of the message words and repeat the process. Once we get the right values (this needs around 2^{23} trials using the difference $d =$ `0x22f80000`) we modify values of M_6, M_{10}, M_{14}, M_{13} and M_2 to adjust the values of $F_3^{(3)}$, $G_3^{(3)}$, $H_3^{(3)}$ to appropriate constants μ_0, μ_1, μ_2. This modification is similar to the original except here we are required to modify M_{13}, whereas the original algorithm avoided it because it was set in branch 4 (instead the original algorithm modified B_0). Now branch 3 is ready.

Branch 4, steps 2–4 We start with choosing random values for M_5, M_1, and M_{15}. Then values of M_8, M_0, and M_{11} are chosen to preserve the subtraction difference d through the first 4 steps of the characteristic. This is easy to do, for

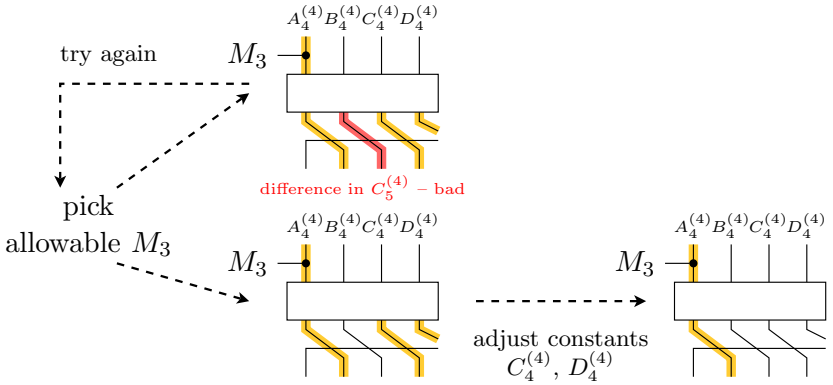


Figure 5.8: Illustration of the procedure used in step 5 of branch 3. We want to get micro-collisions in all three lines without the need for modifying the value of $B_4^{(4)}$.

example, by setting the message blocks so that the input to the f function is zero (the output of the f function is the only thing that can change the subtraction difference). Then we compute up to the beginning of step 5. Next, we use our precomputed table to loop through all choices of M_3 that lead to allowable values and we test each one to see if any of them does not cause a difference propagation to $C_5^{(4)}$ for the current value of $B_4^{(4)}$ that is there. In other words, we are looking for a value of M_3 that actually induces a single micro-collision in line B and has the potential to cause simultaneous micro-collisions in the other two lines. This is illustrated in Fig. 5.8. If no solution is found, then we go back to solve for branch 3 again with new random values.

Once such a solution is found, we have to set the values of $C_4^{(4)}$ and $D_4^{(4)}$ to appropriate constants so that we obtain simultaneous micro-collisions for all three lines. We do this by adjusting the values of M_1, and M_{15} and appropriately compensating for these changes by adjusting M_0 and M_{11}. After this is done, branches 3 and 4 are ready.

Branches 1 and 2 The part of the algorithm that deals with branches 1 and 2 is identical to the one presented in Section 5.7.3 and it does not require any further explanations.

In the original attack from Section 5.7, the search complexity is dominated by branches 1 and 2. There, 2^{64} potential characteristics are obtained for the cost

of 2^{58} FORK-256 operations. Provided that the cost of the modified algorithm for branches 3 and 4 is less than this, the overall complexity is unchanged.

With the difference of $d = \texttt{0x22f80000}$ the probability of passing step 4 of branch 3 is about 2^{-24} and the probability of passing step 5 in branch 4 is about 2^{-19}. The cost of a single check is about eight steps of FORK-256, so 2^{-3} full FORK-256 evaluations. Thus, passing branches 3 and 4 in our modified algorithm requires about 2^{40} FORK-256 evaluations. Hence, it does not influence the final complexity of the attack.

5.8.2 Fixing appropriate chaining values

So far we have removed the need for the fourth initial chaining value to be fixed. This leaves us with three 32-bit words, each one to be set to one of the possible constants required by simultaneous micro-collisions in step 1 of branch 4. This means that by prepending a random message block and computing the digest that in turn becomes the chaining value for the main part of the attack we have the probability of getting the right values of those registers at least 2^{-96}, less than $2^{126.6}$ required for the second phase. However, we can do much better when we use the fact that *any* of the possible constants will suffice in each of the three initial registers.

Let $\mathcal{A}$ be the set of allowable values for g - δ_{15} - f micro-collision in step 1 of branch 4 for a given difference d. For each allowable value $a \in \mathcal{A}$ we can compute sets $\mathcal{F}_a$, $\mathcal{G}_a$, $\mathcal{H}_a$ of constants that yield a micro-collision in the corresponding line. Then, the probability that a randomly selected triple constitute good constants for some allowable value a is

$$P = 1 - \prod_{a \in \mathcal{A}} \left(1 - \frac{|\mathcal{F}_a| \cdot |\mathcal{G}_a| \cdot |\mathcal{H}_a|}{2^{96}}\right)$$

This probability depends on the choice of the difference d. For both differences $d = \texttt{0xdd080000}$ and $d = \texttt{0x22f80000}$ it is equal to $P = 2^{-64.8}$, but there are other differences with much higher values of P. Of course those differences may give worse performance in the main part of the attack because they are not tuned to yield optimal chance of passing requirements of branch 1. What really matters though is that the original differences are suitable for the improved attack.

5.8.3 Experimental results

We implemented this modified strategy and tested it. As an example, we present in Table 5.11 a pair of messages that give a near-collision of weight 42 of the full hash function FORK-256. Here we used difference $d =$`0x3f6bf009` since it has $P = 2^{-21.7}$ for the first phase of the attack.

Table 5.11: Example of a near-collision of weight 42 for the complete hash function FORK-256. The first block is used to obtain the desired values of chaining registers that enable the attack on the compression function.

M	2d4458a4 57976f57 3e44cfd9 1ab54cb2 7ec11870 173f6573 6141c261 7db20d3e 2feeb74d 5fac87a6 61a73fa1 3454b23d 451d389b 78f061ec 7c32fb06 57ef1928 79dcd071 39dc97f0 3a1bff42 031d364c fef000e6 40873ef5 d0741256 649430cf 97ef5538 3eab6a7e b4f9cf72 9eba8257 4e84d457 5a6c49b6 ad1d9711 0f69afa2
M'	2d4458a4 57976f57 3e44cfd9 1ab54cb2 7ec11870 173f6573 6141c261 7db20d3e 2feeb74d 5fac87a6 61a73fa1 3454b23d 451d389b 78f061ec 7c32fb06 57ef1928 79dcd071 39dc97f0 3a1bff42 031d364c fef000e6 40873ef5 d0741256 649430cf 97ef5538 3eab6a7e b4f9cf72 9eba8257 8df0c460 5a6c49b6 ad1d9711 0f69afa2
diff	00000000 83480012 32b4070c 681a1279 648600ad 00000000 00000000 00000000

5.9 Summary

In this chapter we exposed a number of weaknesses of the hash function FORK-256. We showed how the unexpected property of Q-structures (allowing for finding micro-collisions) can be exploited to easily find pairs of messages that after compressing result in a difference on only a small number of bits. We presented how this ease of finding output differences restricted to at most 108 bits may be exploited further to launch a collision-finding attack on the compression function faster and with smaller memory requirements than by birthday attack.

We also showed how the attack on the compression function can be modified to apply to the full hash function.

Our results are by no means final and complete. We expect that having more computational power to search for more favourable cases or investigating slightly different variations of the attacks we presented, it may be possible to improve them significantly.

We believe that apart from the weakness of the step transformation that enabled us to conduct our attack, the structure based on four very short branches

computed in parallel may not be sufficiently secure. The intuition behind this view suggests that for a small number of rounds it is relatively easy to find ways of managing the propagation of differences and separate branches make this process more independent and thus more feasible.

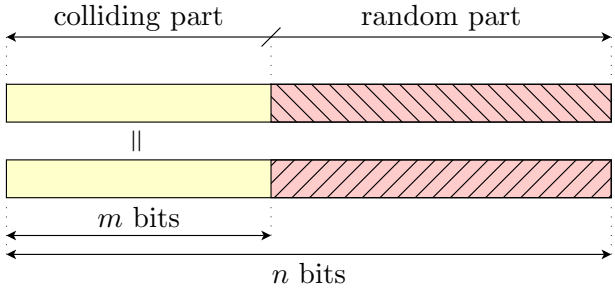
6

Analysis of partially colliding hashes

In this chapter we consider a generic collision-finding problem inspired by our analysis of the hash function FORK-256 (cf. Chapter 5). We ask about the complexity of finding collisions when hash digests are generated in pairs in such a way that for each pair, some fixed part of both digests is identical. For example, the cryptanalysis results of FORK-256 given in Chapter 5 have the first register, part of the second register, and the last three registers equal for each pair that is generated. The other registers, where differences can exist, seem to have the property that the bit differences behave very close to random.

We start with formalising the problem in section 6.1 and then develop a theoretical formula for the probability of collisions in a model using large storage, similar to the classical birthday attack. In section 6.1.4 we compare the result with the classical birthday bound on one hand and the trivial memoryless bound on the other. We generalise our results from pairs to w-tuples in section 6.2, including a comparison and discussion of all three methods (large memory, memoryless, and generic birthday attack). Our conclusion is given in section 6.3.

Figure 6.1: Each partially colliding pair of n-bit values has m bit positions where the pairs are guaranteed to be identical, and $n-m$ bit positions which may differ.



6.1 Theoretical model for partial collisions

6.1.1 The problem

Consider the special case of generating pairs of n-bit hash values that agree on a fixed m-bits and can disagree on the remaining $(n-m)$-bits. We will call these sections of bits the *colliding bits* and the *difference bits*, respectively. The pairs of hash values will be called *partially colliding pairs*. This situation is presented in Fig. 6.1. Assume that the difference bits are uncorrelated and random.

To get an idea of the goal, first consider the "memoryless" version where we examine a single pair at a time, discarding it if there is no collision, and repeating until we find a collision. Since the probability of a collision for each single pair is 2^{m-n} as $n-m$ random bits have to agree, we get that the probability that none of x pairs result in a collision is $(1-2^{m-n})^x$. If we ask for how many pairs x we need in order to have a collision with probability q $(0<q<1)$, we only need to solve $1-(1-2^{m-n})^x = q$. Using the approximation $1-z \approx e^{-z}$ with $z = 2^{m-n}$, which is very accurate when z is small, the solution is $x = -2^{n-m}\ln(1-q)$. For instance, setting $q = 1-\frac{1}{e} \approx 0.63$ and $m = n/2$ the solution is $x = 2^{n/2}$ partially colliding pairs, or $2^{n/2+1}$ hash values total. In comparison to a generic birthday attack, using similar techniques it can be shown (see section 6.1.4) that one needs about $2^{n/2+0.5}$ hash values and a large amount of memory[1] before a collision can be found with probability $1-\frac{1}{e}$. If we ignore the memory issue and ask only which

[1]There are various tradeoffs between memory requirements and pluralisation in differing birthday attack strategies. It practice, one would use a method that has perfect pluralisation. The best such solution seems to be the distinguished point method of [146]. Although it does have reduced memory requirements, it still uses an exponential amount of disk space.

one is faster, then the answer depends upon the time to generate the partially colliding pairs and also the time to find the collision for the birthday strategy once enough pairs are generated. But, since the birthday strategy involves large amounts of memory, one could ask if the use of partially colliding pairs can be improved by also using large amounts of memory.

At first glance, one may carelessly think that a birthday strategy applied to the partially colliding pairs may lead to a run time of order $2^{(n-m)/2}$. Unfortunately this does not work because the colliding bits for pair (a_i, b_i) will, in general, not match the colliding bits for (a_j, b_j), $j \neq i$. But if one keeps all such pairs in memory, in some cases there will be matches. The purpose of this note is to explore how much advantage one can get from these coincidences. We show for the above example that one only needs about $2^{n/2}$ total digests if the memory is available. We also consider generalisations of this idea and explore to what extent large memory can be of benefit.

We have one last remark before we begin – about $1 - z \approx e^{-z}$ and other approximations that we will use. Similar approximations are used in [30]. Ideally, we should present mathematically rigorous arguments bounding the error probability for our application. Unfortunately, bounds such as those given in [9] are not tight enough to draw any meaningful conclusions for the partial collision analysis. Since we do not know how to get mathematically rigorous and meaningful bounds, we instead back up our arguments with experimental evidence. Our computer simulations match the derived formulae quite accurately.

6.1.2 A simple formula approximating the probability

To derive the formula for the probability of a collision, we model our situation as the following occupancy problem

- Boxes correspond to common parts of pairs (colliding parts). There are M boxes ($M = 2^m$)

- We throw x pairs of balls into the boxes. Each ball is selected uniformly from the set of B values ($B = 2^{n-m}$, $n - m$ is the length of the random part of bit strings).

- What is the probability of getting a collision (two identical balls in one box)?

The probability that the box number i contains exactly k pairs is described as the probability of k successes in x Bernoulli trials with success probability $1/M$:

$$Pr[X_i = k] = \binom{x}{k} \left(\frac{1}{M}\right)^k \left(1 - \frac{1}{M}\right)^{x-k} \quad . \tag{6.1}$$

Poisson's theorem [139] tells us that when x tends to infinity in such a way that $x \cdot \frac{1}{M} \to \lambda > 0$, this probability converges to

$$\pi_k = \frac{\lambda^k e^{-\lambda}}{k!}.$$

Moreover, considering the rapidity of the convergence given by the Prohorov's inequality [139, § 7 of Chapter III] stating that $\sum_{k=0}^{\infty} |Pr[X_i = k] - \pi_k| \leq \frac{2\lambda}{x} \cdot \min(2, \lambda)$ we can safely assume that

$$Pr[X_i = k] \approx \frac{1}{k!}\lambda^k e^{-\lambda} \quad . \tag{6.2}$$

where $\lambda = x/M$.

The expected number of bins containing exactly k pairs is equal to

$$E_k = M \cdot P[X_i = k] \approx \frac{M}{k!}\lambda^k e^{-\lambda} \quad . \tag{6.3}$$

If a container holds k pairs, then there are $2k$ balls altogether. There is no collision in that particular container if all balls are different. This happens with probability

$$\hat{p}_k = \left(1 - \frac{1}{B}\right)\left(1 - \frac{2}{B}\right)\cdots\left(1 - \frac{2k-1}{B}\right) = \prod_{j=1}^{2k-1}\left(1 - \frac{j}{B}\right) \approx$$
$$\prod_{j=1}^{2k-1} e^{-j/B} = \mathrm{Exp}\left[-\frac{1}{B}\sum_{j=1}^{2k-1} j\right] = \mathrm{Exp}\left[-\frac{2k(2k-1)}{2B}\right] = e^{-\frac{k(2k-1)}{B}} \quad . \tag{6.4}$$

Taking into account all E_k boxes containing k pairs, the probability that no collision is present in any of the boxes is about

$$\bar{p}_k = \hat{p}_k^{E_k} = \left(e^{-k(2k-1)/B}\right)^{\frac{M}{k!}\lambda^k e^{-\lambda}} = \mathrm{Exp}\left[-\frac{k(2k-1)}{B}\cdot\frac{M}{k!}\lambda^k e^{-\lambda}\right] = \mathrm{Exp}\left[-\frac{M}{B}\cdot e^{-\lambda}\frac{2k-1}{(k-1)!}\lambda^k\right] \quad . \tag{6.5}$$

Now, the probability that there is no collision at all is the product of probabilities $\bar{p}_k$ taken over all values of k up to the limit of x.

$$\bar{P} = \prod_{k=1}^{x} \mathrm{Exp}\left[-\frac{M}{B}\cdot e^{-\lambda}\frac{2k-1}{(k-1)!}\lambda^k\right] = \mathrm{Exp}\left[-M/B\cdot e^{-\lambda}\sum_{k=1}^{x}\frac{2k-1}{(k-1)!}\lambda^k\right] \tag{6.6}$$

So we end up with the final probability of a collision in such a scenario approximately equal to

$$P = 1 - \mathrm{Exp}\left[-\frac{M}{B}\cdot e^{-\lambda}\sum_{k=1}^{x}\frac{2k-1}{(k-1)!}\lambda^k\right] \quad .$$

In our case $M = 2^m$ and $B = 2^{n-m}$, so $M/B = 2^{2m-n}$ and

$$P = 1 - \mathrm{Exp}\left[-2^{2m-n}\cdot e^{-\lambda}\sum_{k=1}^{x}\frac{2k-1}{(k-1)!}\lambda^k\right] \quad . \tag{6.7}$$

We can simplify (6.7) further using the fact that $\sum_{k=0}^{\infty} z^k/k! = e^z$ which converges very fast, so $\sum_{k=0}^{x} z^k/k! \approx e^z$ is extremely accurate. Hence, the summation in the exponent in (6.7) becomes

$$\begin{aligned}\sum_{k=1}^{x}\frac{2k-1}{(k-1)!}\lambda^k &= \sum_{k=0}^{x-1}\frac{2k+1}{k!}\lambda^{k+1} = \lambda\left(2\sum_{k=0}^{x-1}\frac{k}{k!}\lambda^k + \sum_{k=0}^{x-1}\frac{1}{k!}\lambda^k\right)\\ &= \lambda\left(2\sum_{k=1}^{x-1}\frac{k}{k!}\lambda^k + \sum_{k=0}^{x-1}\frac{1}{k!}\lambda^k\right) = \lambda\left(2\sum_{k=1}^{x-1}\frac{1}{(k-1)!}\lambda^k + \sum_{k=0}^{x-1}\frac{1}{k!}\lambda^k\right)\\ &= \lambda\left(2\lambda\sum_{k=0}^{x-2}\frac{1}{k!}\lambda^k + \sum_{k=0}^{x-1}\frac{1}{k!}\lambda^k\right) \approx \lambda\left(2\lambda e^{\lambda} + e^{\lambda}\right) = \lambda e^{\lambda}(2\lambda+1) \quad .\end{aligned} \tag{6.8}$$

Plugging in (6.8) to (6.7) we get the estimation

$$P = 1 - \mathrm{Exp}\left[-2^{2m-n}\cdot\lambda(2\lambda+1)\right]$$

and since $\lambda = x/M = x \cdot 2^{-m}$ we can rewrite the probability of collision as

$$P = 1 - \mathrm{Exp}\left[-2^{m-n} \cdot x - 2^{1-n} \cdot x^2\right] \quad . \tag{6.9}$$

6.1.3 Practical verification of the formula

We wrote a small program that simulates the occupancy problem in question and compared the experimental results with those given by formula (6.9). Experimental probabilities are computed as a fraction of tests that yield collision to the total number of tests equal 100000. The results presented in Table 6.1 match very well values obtained by the means of formula (6.9). Below we have $N = 2^n$ and $M = 2^m$.

Table 6.1: Comparison of probabilities of collisions obtained by a simulation with those computed using our formula. Values are rounded to 4 decimal places.

$N = 2^{20}$ $M = 2^8$							
$x = 2^q$, $q =$	5	6	7	8	9	10	11
Pr experiment	0.0097	0.0225	0.0608	0.1694	0.4631	0.8955	0.9999
Pr theoretical	0.0097	0.0232	0.0606	0.1710	0.4647	0.8946	0.9998

$N = 2^{20}$ $M = 2^{10}$							
$x = 2^q$, $q =$	5	6	7	8	9	10	11
Pr experiment	0.0331	0.06795	0.1446	0.3136	0.6302	0.9528	0.9999
Pr theoretical	0.0326	0.06790	0.1447	0.3127	0.6321	0.9502	0.9999

$N = 2^{20}$ $M = 2^{12}$							
$x = 2^q$, $q =$	6	7	8	9	10	11	12
Pr experiment	0.22917	0.4149	0.6750	0.9189	0.9976	1.0000	1.0000
Pr theoretical	0.22726	0.4121	0.6753	0.9179	0.9975	1.0000	1.0000

$N = 2^{24}$ $M = 2^{10}$							
$x = 2^q$, $q =$	7	8	9	10	11	12	13
Pr experiment	0.0095	0.0223	0.0592	0.1710	0.4656	0.8946	0.9997
Pr theoretical	0.0097	0.0232	0.0606	0.1710	0.4647	0.8946	0.9998

$N = 2^{24}$ $M = 2^{12}$							
$x = 2^q$, $q =$	7	8	9	10	11	12	13
Pr experiment	0.0329	0.0670	0.1434	0.3135	0.6312	0.9496	0.99996
Pr theoretical	0.0326	0.0679	0.1446	0.3127	0.6321	0.9502	0.99995

6.1.4 How much do we gain?

We now answer the core question how much data can be saved compared to the generic birthday attack if we can generate such special pairs of outputs with restricted differences, and also assuming that all data can fit in memory.

In the classical birthday paradox attack, the probability that we have a collision after storing x_{brt} values is equal to

$$P_{brt} = 1 - \prod_{k=0}^{x_{brt}-1} (1 - \frac{k}{2^n}) \approx 1 - \mathrm{Exp}\left[-\frac{1}{2^n} \sum_{k=0}^{x_{brt}-1} k\right] = 1 - \mathrm{Exp}\left[-\frac{x_{brt}(x_{brt}-1)}{2^{n+1}}\right]$$

If we want the probability of a collision to be q, solving for x_{brt} we get the result

$$x_{brt} \approx \sqrt{-2^{n+1} \ln(1-q)} \ .$$

For $q = 1 - \frac{1}{e}$, this is $2^{n/2+0.5}$.

On the other hand, if we generate pairs with restricted differences, plugging in $P = q$ into (6.9) gives the following quadratic equation

$$2 \cdot x^2 + 2^m \cdot x - 2^n \ln(1-q) = 0 \ .$$

There are two solutions to this equation, one positive and one negative. Only the positive solution makes sense in our context so the unique answer is

$$x = \frac{1}{4}\left(-2^m + \sqrt{2^{2m} - 2^n \cdot 8 \ln(1-q)}\right)$$

pairs of values (so the total number of digests is twice this).

Considering the case when $m = n/2$ and $q = 1 - \frac{1}{e}$, we get $2^{n/2-1}$ pairs of hash digests, or $2^{n/2}$ digests total. This is a factor of $\sqrt{2}$ times better than the classical birthday attack.

6.1.5 Discussion

A summary of the derived formulae is given in Table 6.2. Observe that the number of hash digests for the memoryless partial collision strategy is less than that of

Table 6.2: Comparison of the total number of digests need to find a collision with probability q for the three different methods.

method	total number digests general formula	special case $m = n/2$
birthday attack	$2^{n/2+0.5}\sqrt{-\ln(1-q)}$	$2^{n/2+0.5}\sqrt{-\ln(1-q)}$
memoryless partial colls	$-2^{n-m+1}\ln(1-q)$	$-2^{n/2+1}\ln(1-q)$
large memory partial colls	$\frac{1}{2}\left(-2^m + \sqrt{2^{2m} - 2^{n+3}\ln(1-q)}\right)$	$2^{n/2-1}\left(\sqrt{1-8\ln(1-q)}-1\right)$

the birthday strategy whenever $2^{n-2m+1} < -1/\ln(1-q)$. For $q = 1 - 1/e$, this means $m > (n+1)/2$. The memoryless colliding pairs strategy becomes costly as m becomes smaller than half of n, as one would expect since the difference bits occupy more than half of the hash digests. On the other hand, the large memory colliding pairs strategy does not suffer from this because it essentially contains within it a birthday paradox strategy. So it always is at least as good as birthday paradox, though the benefit quickly becomes small as m does.

Another observation is that the large memory version has negligible benefit over the memoryless version as m gets much larger than half of n. Consequently, our results do not make any real improvement over the FORK-256 analysis of [103], since there $m = 148$ and $n = 256$. It is conceivable, however, that the strategy could be applied to some other hash function to get a better than birthday attack bound. In the past, people have sought after differential paths that would lead to full collisions, and then attempt to find data satisfying those paths. Our results suggest that one may consider more general paths that only lead to partial collisions in order to potentially improve (slightly) on birthday attacks. In the next section, we show a generalisation that leads to more promising improvements.

6.2 Generalisation for w-tuples

We can generalise the above results for the scenario where not pairs, but rather tuples of w elements are generated. In this case, assume we are able to generate w digests that all share a common fixed part (the colliding bits) and have random

differences for the remaining part (the difference bits). Intuitively, one might expect that we need less total hash digests for this case because within each w–tuple we have a birthday paradox type phenomenon working for us, though it is a much milder one than the standard birthday paradox. But how much of an advantage we get and how much large memory can help is not clear prior to doing the analysis. The following two subsections does these analyses, and a comparison is given in the third subsection.

6.2.1 Memoryless version

In the "memoryless" version of the attack, we generate a w–tuple of partially colliding hashes and examine to see if there is a collision amongst any two pairs within the tuple. If there is no collision, we throw it away and repeat the process until a collision is finally found.

The probability of no collision within the w–tuple is

$$\prod_{j=1}^{w-1}\left(1-\frac{j}{B}\right) \approx e^{\frac{-w(w-1)}{2B}} \ .$$

The number of w–tuples x needed for probability q of a collision is thus given by the solution to $q = e^{\frac{-w(w-1)}{2B}x}$, which is

$$x = -\frac{2^{n-m+1}}{w(w-1)}\ln(1-q) \ .$$

The total number of hashes computed is $-\frac{2^{n-m+1}}{w-1}\ln(1-q)$.

6.2.2 Large memory version

For the large memory version, we generalise the work of section 6.1. Generalising (6.4) we have:

$$\hat{p}_k = \left(1-\frac{1}{B}\right)\left(1-\frac{2}{B}\right)\cdots\left(1-\frac{wk-1}{B}\right) \approx e^{-\frac{wk(wk-1)}{2B}} \ . \tag{6.10}$$

Following the same line of argument as we used to derive (6.6), we obtain

$$\bar{P} = \mathrm{Exp}\left[-\frac{M}{2B}\cdot e^{-\lambda}\sum_{k=1}^{x}\frac{wk(wk-1)}{k!}\lambda^k\right] \ .$$

Table 6.3: Comparison of numbers of hash digests required to get probability $1 - \frac{1}{e}$ of finding a collision for different values of w when $n = 256$, $m = 128$. A generic birthday attack requires $2^{128.5}$ hash digests.

	total number of hashes	
w	memoryless version	large memory version
2	$2^{129.00}$	$2^{128.00}$
3	$2^{128.00}$	$2^{127.55}$
4	$2^{127.42}$	$2^{127.17}$
8	$2^{126.19}$	$2^{126.14}$
16	$2^{125.09}$	$2^{125.08}$
256	$2^{121.01}$	$2^{121.01}$
2^{16}	$2^{113.00}$	$2^{113.00}$

Again, using the Maclaurin series expansion of the exponential function we obtain a close approximation to the sum $\sum_{k=1}^{x} \frac{wk(wk-1)}{k!} \approx e^{\lambda}\left(w^2\lambda^2 + w^2\lambda - w\lambda\right)$, and use it to get the following generalisation of (6.9)

$$\begin{aligned} P =& 1 - \mathrm{Exp}\left[-\frac{M}{2B} w\lambda(w\lambda + w - 1)\right] = 1 - \mathrm{Exp}\left[-\frac{wx}{2B}\left(\frac{wx}{M} + w - 1\right)\right] \\ =& 1 - \mathrm{Exp}\left[-wx \cdot 2^{-n-1}\left(wx + (w-1)2^m\right)\right] \quad . \end{aligned} \tag{6.11}$$

Once can then derive the formula

$$x = \frac{1}{2w}\left(-(w-1)2^m + \sqrt{(w-1)^2 \cdot 2^{2m} - 8 \cdot 2^n \ln(1-q)}\right)$$

for the number x of w-tuples that have to be stored to assure probability q of a collision. The total number of hash digests is w times this, i.e.

$$\frac{1}{2}\left(-(w-1)2^m + \sqrt{(w-1)^2 \cdot 2^{2m} - 8 \cdot 2^n \ln(1-q)}\right) \quad .$$

6.2.3 Comparisons

Table 6.3 compares the number of total digests needed for both the memoryless and large memory versions with $n = 256$, $m = 128$, $q = 1 - \frac{1}{e}$, and using various choices of w. Recall also that the generic birthday attack that needs $2^{n/2+0.5} = 2^{128.5}$ hashes.

One can readily see that the ability to generate these w–tuples efficiently can

lead to improved attacks on hash functions. This further justifies the potential application of searching for partially colliding differentials (as opposed to fully colliding differentials, which may be harder to find) in the cryptanalysis of hash functions. It also shows that the memoryless version is just as good as the one using large memory for almost all values of w.

In fact, section 3.3 of the analysis of FORK-256 paper [103] presents a similar scenario, with $w = 2^{32}$, $m = 128$, and $n = 256$. However, they are only able to get a 1-bit near collision (since the differential has the property that it can never be a full collision). Moreover, their results (based on experiments) are better than our theory predicts due to the observation that the difference bits seem to be somewhat correlated.

6.3 Summary

In this chapter we presented results of our investigation into partially colliding digests. We demonstrated that the use of partially colliding pairs may lead to better than birthday paradox attacks on hash functions. Our analysis was mainly concerned with the number of digests needed to be generated, and ignored practical memory requirements. Furthermore, the use of partially colliding w–tuples has much more potential, since it can require much less hash digests and does not need large memory in order to get good benefits but the applicability of these results is contingent upon efficient methods of finding partial collisions.

7

Conclusions and future research directions

7.1 Thesis summary

In this thesis we studied problems related to analysis and design of dedicated hash functions. We performed security analysis of a number of popular designs and showed weaknesses in some of them. All our results show that differential cryptanalsis, a fundamental tool used to analyse hash functions, has still a lot of potential when applied to dedicated designs. We showed that the fundamental problem of finding suitable differential characteristics is closely related to a well-known problem in coding theory of finding codewords with minimal weights. We have also illustrated how this observation can be used as the fundamental cryptanalytic tool.

Even though this thesis contains mainly negative results, i.e. we exhibit potiential or actual weaknesses in some designs of dedicated hash functions, we hope that it will also contribute to the design aspect by highlighting the crucial issues that need to be consider during the design process.

Note that differential cryptanalysis is still one of the most effective attacks that can be launched against hash functions.

This work documents a tiny fraction of recent results that show weaknesses of dedicated hash functions. These results show that the main difficulty and the main challenge when designing dedicated cryptographic hash functions is the immunity against differential analysis. However, most of dedicated hash functions designed up to date do not come with any security proof that would guarantee that some of the security gaols are achieved. We believe that this situation should change and designers should turn to some well-established solutions used for block ciphers like the Wide Trail Strategy [43].

7.2 Future research directions

7.2.1 Design

Recent cryptographic attacks that threatened some of the most popular hash functions generated a lot of interest in this area. They also stimulated NIST to announce a competition for the Advanced Hash Standard [117], similar to the one used to select AES. This means that in coming years the research on hash functions will concentrate on the development of a new algorithm (or algorithms) that will be future replacements for functions of the SHA family.

The challenge seems to be greater than for the AES competition. This is mainly because there is no agreement in the cryptographic community on what exactly are the security properties we should expect from the new hash standard.

Therefore the first task is to determine the required security properties of the hash functions in the context of their present and future applications.

Only after that stage is finished, researchers can focus on designing functions that satisfy those properties.

While designing hash functions, there are two related although different aspects, namely, the design of secure compression functions and the design of appropriate modes of operation that can transfer the security properties from the underlying compression function to the hash function.

Heuristic designs of hash functions do not guarantee that the resulting hash

achieves the appropriate security goals. The way out could be to design provably secure hash functions. In such designs, breaking the security goals of hash function is equivalent to breaking a well-known intractable problem (such as factorization or discrete logarithm).

It seems that to avoid unpleasant surprises that are always possible for heuristic designs, the preferred direction of research would be into provably secure designs. A lot of research is awaiting in this direction, most notably on improving the efficiency of provably secure constructions and finding new, suitable hard problems.

In a longer perspective, one should take into account the growing potential of quantum computing that can significantly change our view of intractable problems.

So far, the field of quantum computing is only beginning to emerge, but one has to bear in mind that in 30 years quantum computers may become practical and this potential threat should be also considered for applications requiring very long life.

7.2.2 Analysis

A possible continuation of the research that is the topic of this thesis could go in two directions. Authors of FORK-256 recently presented an improved version of the design [80]. They used bijective functions f and g and changed the step transformation to elliminate the possibility of simultaneous micro-collisions. Still, the design is based on four short parallel branches and it is an interesting question to see whether this construction is significantly more secure than the previous one.

Another, very interesting question is how to get better attacks on SHA-256. It is definitely worth looking at the possiblity of adopting the method presented in [50] to the structure of SHA-256. It involves many substantial difficulties, like the problem of a very complex nature of step transformation, complicated nonlinear relations in the step transformation and the problem of finding a message that would follow the characteristics that works in spite of limited freedom in adjusting particular values of bits by changing message bits. In this context, it may be worth investingating tools such as SAT solvers, recently applied to

analysis of hash functions [113, 48].

Finally, the hash function competition is going to be a really exciting event for cryptanalysts. We can expect a host of new, interesting designs, a variety of fresh approaches and solutions. It will be a rich source of designs that need testing and evaluation, the favourite work of a cryptanalyst. I am really looking forward to it.

Bibliography

[1] —. The fasttrack protocol. web page, `http://gnunet.org/papers/FAST-TRACK-PROTOCOL`. accessed 29/05/2007.

[2] —. RC5-64 has been solved! `http://www1.distributed.net/pressroom/news-20020926.txt`, September 25 2002.

[3] R. Anderson and E. Biham. Tiger: A fast new hash function. In D. Gollmann, editor, *Fast Software Encryption – FSE'96*, volume 1039 of *LNCS*, pages 121–144. Springer-Verlag, 1996.

[4] D. Augot, M. Finiasz, and N. Sendrier. A family of fast syndrome based cryptographic hash functions. In *Progress in Cryptology – Mycrypt 2005*, volume 3715 of *LNCS*, pages 64–83. Springer, 2005.

[5] M. Bellare. New proofs for NMAC and HMAC: Security without collision-resistance. In *Advances in Cryptology — CRYPTO'06*, volume 4117 of *LNCS*, pages 602–619. Springer, 2006.

[6] M. Bellare, A. Boldyreva, and A. Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 171–188. Springer, 2004.

[7] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology – CRYPTO'96*, volume 1109 of *LNCS*, pages 1–15. Springer, 1996.

[8] M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: The case of hashing and signing. In *Advances in Cryptology – CRYPTO '94*, volume 839 of *LNCS*, pages 216–233. Springer, 1994.

[9] M. Bellare and T. Kohno. Hash function balance and its impact on birthday attacks. In *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 401–418. Springer, 2004.

[10] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proc. First ACM Conference on Computer and Communications Security*, pages 62–73, Fairfax, 1993. ACM.

[11] M. Bellare and P. Rogaway. Optimal asymmetric encryption – how to encrypt with RSA. In *Advances in Cryptology – EUROCRYPT '94*, volume 950 of *LNCS*, pages 92–111. Springer, 1994.

[12] M. Bellare and P. Rogaway. The exact security of digital signatures - how to sign with RSA and Rabin. In *Advances in Cryptology – EUROCRYPT '96*, volume 1070 of *LNCS*, pages 399–416. Springer, 1996.

[13] E. Biham and R. Chen. Near collisions of SHA-0. In M. Franklin, editor, *Advances in Cryptology – CRYPTO'04*, volume 3152 of *LNCS*. Springer-Verlag, 2004.

[14] E. Biham and R. Chen. New results on SHA-0 and SHA-1. Presented at the rump session of CRYPTO'04, 2004.

[15] E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, and W. Jalby. Collisions of SHA-0 and reduced SHA-1. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT'05*, volume 3494, pages 36–57. Springer-Verlag, 2005.

[16] E. Biham and O. Dunkelman. A framework for iterative hash functions – HAIFA. 2nd NIST Hash Workshop, August 2006. available from `http://csrc.nist.gov/pki/HashWorkshop/2006/program_2006.htm`.

[17] E. Biham and O. Dunkelman. A framework for iterative hash functions – HAIFA. Technical Report CS-2007-15, Technion – Israel Institute of Technology, 2007. Available from `http://www.cs.technion.ac.il/tech-reports/`.

[18] E. Biham, O. Dunkelman, and N. Keller. Rectangle attacks on 49-round SHACAL-1. In *Fast Software Encryption – FSE '03*, volume 2887 of *LNCS*, pages 22–35. Springer, 2003.

[19] E. Biham, O. Dunkelman, and N. Keller. A simple related-key attack on the full SHACAL-1. In *Topics in Cryptology – CT-RSA 2007*, volume 4377 of *LNCS*, pages 20–30. Springer, 2007.

[20] O. Billet, T. Peyrin, and M. J. Robshaw. On building hash functions from multivariate quadratic equations. In *12th Australasian Conference on Information Security and Privacy, ACISP 2007*, LNCS. Springer, 2007. to appear.

[21] J. Black, M. Cochran, and T. Highland. A study of the MD5 attacks: Insights and improvements. In *Fast Software Encryption, FSE'06*, volume 4047 of *LNCS*, pages 262–277. Springer, 2006.

[22] J. Black, M. Cochran, and T. Shrimpton. On the impossibility of highly-efficient blockcipher-based hash functions. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 526–541. Springer-Verlag, 2005.

[23] J. Black, P. Rogaway, and T. Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In *Advances in Cryptology – CRYPTO '02*, volume 2442 of *LNCS*, pages 103–118. Springer, 2002.

[24] B. d. Boer and A. Bosselaers. An attack on the last two rounds of MD4. In J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO'91*, volume 576 of *LNCS*, pages 194–203. Springer-Verlag, 1991.

[25] B. d. Boer and A. Bosselaers. Collisions for the compression function of MD5. In *Advances in Cryptology – EUROCRYPT '93*, volume 765 of *LNCS*, pages 293–304. Springer-Verlag, 1994.

[26] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system I: The user language. *Journal of Symbolic Computation*, 24(3–4):235–265, 1997. `http://magma.maths.usyd.edu.au/`.

[27] A. Bosselaers and B. Preneel, editors. *Integrity Primitives for Secure Information Systems. Final Report of RACE Integrity Primitives Evaluation.*, volume 1007 of *LNCS*. Springer-Verlag, 1995.

[28] R. Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *Advances in Cryptology – CRYPTO'97*, volume 1294 of *LNCS*, pages 455–469. Springer, 1997.

[29] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[30] J. Canny. Occupancy problems – CS174 course lecture notes. `http://www.cs.berkeley.edu/~fc/cs174/lecs/lec5/lec5.pdf`, 2001.

[31] A. Canteaut and F. 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.

[32] F. Chabaud. On the security of some cryptosystems based on error-correcting codes. In A. D. Santis, editor, *Advances in Cryptology - EuroCrypt '94*, volume 950 of *Lecture Notes in Computer Science*, pages 131–139, Berlin, 1995. Springer-Verlag.

[33] F. Chabaud and A. Joux. Differential collisions in SHA-0. In H. Krawczyk, editor, *Advances in Cryptology - CRYPTO'98*, volume 1462 of *LNCS*, pages 56–71. Springer-Verlag, 1998.

[34] R. Chatterjee, M. Saifee, and D. RoyChowdhury. Modifications of SHA-0 to prevent attacks. In *Information Systems Security – ICISS'05*, volume 3803 of *LNCS*, pages 277–289. Springer, 2005.

[35] H.-S. Cho, S. Park, S. Sung, and A. Yun. Collision search attack for 53-step HAS-160. In *Information Security and Cryptology – ICISC 2006*, volume 4296 of *LNCS*, pages 286–295. Springer, 2006.

[36] S. Contini, A. K. Lenstra, and R. Steinfeld. VSH, an efficient and provable collision-resistant hash function. In *Advances in Cryptology – EUROCRYPT '06*, volume 4004 of *LNCS*, pages 165–182. Springer, 2006.

[37] S. Contini, K. Matusiewicz, and J. Pieprzyk. Extending FORK-256 attack to the full hash function. In *Proc. ICICS 2007*, volume 4861 of *LNCS*, pages 296–305. Springer, Dec 2007.

[38] S. Contini, R. Steinfeld, J. Pieprzyk, and K. Matusiewicz. A critical look at cryptographic hash function literature. ECRYPT Hash Workshop 2007, May, 24–25 2007. available from `http://events.iaik.tugraz.at/HashWorkshop07/program.html`.

[39] S. Contini and Y. L. Yin. Forgery and partial key-recovery attacks on HMAC and NMAC using hash collisions. In *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 37–53. Springer, 2006.

[40] J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-damgård revisited: How to construct a hash function. In *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *LNCS*, pages 430–448. Springer, 2005.

[41] S. A. Crosby and D. S. Wallach. Denial of service via algorithmic complexity attacks. In *Proc. 12th USENIX Security Symposium*, pages 29–44, 2003.

[42] J. Daemen. *Cipher and Hash Function Design, Strategies based on linear and differential cryptanalysis.* PhD thesis, Katholieke Universiteit Leuven, 1995.

[43] J. Daemen and V. Rijmen. The wide trail design strategy. In *Cryptography and Coding: 8th IMA International Conference*, volume 2260 of *LNCS*, pages 222–238. Springer, 2001.

[44] I. B. Damgård. Collision free hash functions and public key signature schemes. In *Advances in Cryptology – EUROCRYPT '87*, volume 304 of *LNCS*, pages 203–216, 1988.

[45] I. B. Damgård. A design principle for hash functions. In G. Brassard, editor, *Advances in Cryptology - CRYPTO '89*, volume 435 of *LNCS*, pages 416–427. Springer-Verlag, 1989.

[46] M. Daum. *Cryptanalysis of Hash Functions of the MD4-Family.* PhD thesis, Ruhr-Universität Bochum, May 2005.

[47] G. Davida. Chosen signature cryptanalysis of the RSA (MIT) public key cryptosystem. Technical report, Dept. of Electrical Engineering and Computer Science, University of Wisconsin-Milwaukee, 1982.

[48] D. De, A. Kumarasubramanian, and R. Venkatesan. Inversion attacks on secure hash functions using SAT solvers. In *Theory and Applications of Satisfiability Testing – SAT 2007*, volume 4501 of *LNCS*, pages 377–382. Springer, 2007.

[49] C. De Canniére, F. Mendel, and C. Rechberger. A collision for 70-step SHA-1 in a minute. Presented at the rump session of FSE'07, March 2007. `http://fse2007.uni.lu/slides/rump/sha.pdf`.

[50] C. De Canniére and C. Rechberger. Finding SHA-1 characteristics: General results and applications. In *Advances in Cryptology – ASIACRYPT'06*, volume 4284 of *LNCS*, pages 1–20. Springer, Dec 2006.

[51] R. D. Dean. *Formal Aspects of Mobile Code Security*. PhD thesis, Princeton University, 1999. available from `http://www.cs.princeton.edu/sip/pub/ddean-dissertation.php3`.

[52] D. E. Denning. Digital signatures with RSA and other public-key cryptosystems. *Commun. ACM*, 27(4):388–392, 1984.

[53] A. Desai, A. Hevia, and Y. L. Yin. A practice-oriented treatment of pseudorandom number generators. In *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 368–383. Springer, 2002.

[54] Y. G. Desmedt and A. M. Odlyzko. A chosen text attack on the RSA cryptosystem and some discrete logarithm schemes. In *Advances in Cryptology – CRYPTO '85*, volume 218 of *LNCS*, pages 516–522. Springer, 1985.

[55] H. Dobbertin. Cryptanalysis of MD4. In *Fast Software Encryption – FSE '96*, volume 1039 of *LNCS*, pages 53–69. Springer-Verlag, 1996.

[56] H. Dobbertin. Cryptanalysis of MD5. Presented at the rump session of EUROCRYPT '96, May 12-16, 1996.

[57] H. Dobbertin. The status of MD5 after a recent attack. *CryptoBytes*, 2(2):1,3–6, 1996.

[58] H. Dobbertin. RIPEMD with two-round compress function is not collision-free. *Journal of Cryptology*, 10(1):51–70, 1997.

[59] H. Dobbertin. Cryptanalysis of MD4. *Journal of Cryptology*, 11(4):253–271, 1998.

[60] H. Dobbertin. The first two rounds of MD4 are not one-way. In *Fast Software Encryption – FSE '98*, volume 1372 of *LNCS*, pages 284–292. Springer, 1998.

[61] Y. Dodis, R. Gennaro, J. Håstad, and T. Krawczyk, Hugo AU Rabin. Randomness extraction and key derivation using the CBC, cascade and HMAC modes. In *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *LNCS*, pages 494–510. Springer, 2004.

[62] O. Dunkelman, N. Keller, and J. Kim. Related-key rectangle attack on the full SHACAL-1. In *Selected Areas in Cryptology – SAC '06*, LNCS. Springer, 2006. to appear.

[63] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO '86*, volume 263 of *LNCS*, pages 186–194. Springer, 1986.

[64] D. L. G. Filho, P. S. L. M. Barreto, and V. Rijmen. The Maelstrom-0 hash function. In *Proc. VI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, September 2006. available from `http://www.cryptolounge.org/w/images/f/f5/Maelstrom-0.pdf`.

[65] FIPS 180. Secure hash standard (SHS). National Institute of Standards and Technology, May 1993. Replaced by [116].

[66] FIPS PUB 186-2. Digital signature standard. National Institute of Standards and Technology, 1994.

[67] FIPS PUB 186-2 (Change Notice 1). Digital signature standard. National Institute of Standards and Technology, 2001.

[68] P. Gauravaram, W. Millan, E. Dawson, and K. Viswanathan. Constructing secure hash functions by enhancing merkle-damgård construction. In *Information Security and Privacy – ACISP 2006*, volume 4058 of *LNCS*, pages 407–420. Springer, 2006.

[69] M. Gebhardt, G. Illies, and W. Schindler. A note on the practical value of single hash collisions for special file formats. In J. Dittmann, editor, *Sicherheit 2006*, volume P-77 of *LI*, pages 333–344. Gesellschaft für Informatik e.v., 2006. `http://www.gi-ev.de/fileadmin/redaktion/2006_LNI/LNI-P-77.pdf`.

[70] J. Gibson. Discrete logarithm hash function that is collision free and one way. *IEE Proceedings-E*, 138(6):407–410, 1991.

[71] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of ACM*, 33(4):792–807, 1986.

[72] Z. Gutterman, B. Pinkas, and T. Reinman. Analysis of the Linux random number generator. Cryptology ePrint Archive, Report 2006/086, 2006. available from `http://eprint.iacr.org/2006/086/`.

[73] S. Halevi and H. Krawczyk. Strengthening digital signatures via randomized hashing. In *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *LNCS*, pages 41–59. Springer, 2006.

[74] H. Handschuch and D. Naccache. SHACAL. Submission to the NESSIE project, 2000. Available from `www.cryptonessie.org`.

[75] H. Handschuh, L. R. Knudsen, and M. J. Robshaw. Analysis of SHA-1 in encryption mode. In *Progress in Cryptology – CT-RSA 2001*, volume 2020 of *LNCS*, pages 70–83. Springer-Verlag, 2001.

[76] P. Hawkes, M. Paddon, and G. Rose. Automated search for round 1 differentials for SHA-1: Work in progress. Presented at 2nd NIST Hash Workshop, August 2006. Available from `http://www.csrc.nist.gov/pki/HashWorkshop/2006/program_2006.htm`.

[77] P. Hawkes, M. Paddon, and G. G. Rose. Musings on the wang et al. MD5 collision. Cryptology ePrint Archive, Report 2004/264, Oct 2004. `http://eprint.iacr.org/2004/264`.

[78] P. Hawkes, M. Paddon, and G. G. Rose. On corrective patterns for the SHA-2 family. Cryptology ePrint Archive, Report 2004/207, August 2004. `http://eprint.iacr.org/`.

[79] Y.-S. Her and K. Sakurai. On the security of yet another reduced version of 3-pass HAVAL. In *Proc. IEEE International Symposium on Information Theory 2003*, 2003.

[80] D. Hong, D. Chang, J. Sung, S. Lee, S. Hong, J. Lee, D. Moon, and S. Chee. New FORK-256. Cryptology ePrint Archive, Report 2007/185. `http://eprint.iacr.org/2007/185`.

[81] D. Hong, J. Sung, S. Hong, S. Lee, and D. Moon. A new dedicated 256-bit hash function: FORK-256. First NIST Workshop on Hash Functions, 2005.

[82] D. Hong, J. Sung, S. Lee, D. Moon, and S. Chee. A new dedicated 256-bit hash function. In *Fast Software Encryption – FSE'06*, LNCS. Springer-Verlag, 2006.

[83] A. Joux. Collisions in SHA-0. Short talk presented at CRYPTO'04 Rump Session, 2004.

[84] A. Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *LNCS*, pages 306–316. Springer, 2004.

[85] C. S. Jutla and A. C. Patthak. Provably good codes for hash function design. In *Selected Areas in Cryptography – SAC'06*. Springer, 2006. to appear, available from `http://www.cs.utexas.edu/~anindya/`.

[86] R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

[87] P. Kasselman and Penzhorn. Cryptanalysis of reduced version of HAVAL. *IEE Electronics Letters*, 36(1):30–31, Jan 2000.

[88] J. Kelsey and B. Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In R. Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 474–490. Springer-Verlag, 2005.

[89] J. Kim, A. Biryukov, B. Preneel, and S. Lee. On the security of encryption modes of MD4, MD5 and HAVAL. In *International Conference on Information and Communications Security – ICICS 2005*, volume 3783 of *LNCS*, pages 147–158. Springer, 2005.

[90] J. Kim, D. Moon, W. Lee, S. Hong, S. Lee, and S. Jung. Amplified boomerang attack against reduced-round SHACAL. In *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 829–835. Springer, 2002.

[91] V. Klima. Finding MD5 collisions - a toy for a notebook. Cryptology ePrint Archive, Report 2005/075, 2005. `http://eprint.iacr.org/`.

[92] V. Klima. Finding MD5 collisions on a notebook PC using multi-message modifications. Cryptology ePrint Archive, Report 2005/102, 2005. `http://eprint.iacr.org/2005/102`.

[93] V. Klima. Tunnels in hash functions: MD5 collisions within a minute. Cryptology ePrint Archive, Report 2006/105, 2006. http://eprint.iacr.org/2006/105.

[94] D. E. Knuth. *The Art of Computer Programming. Volume 3, Sorting and Searching*. Addison-Wesley, 2005.

[95] C. Lemuet. Collision in SHA-0. sci.crypt newsgroup message, Message-ID: cfg007$1h1b$1@io.uvsq.fr, 12 August 2004.

[96] A. Lenstra and B. de Weger. On the possibility of constructing meaningful hash collisions for public keys. In *Information Security and Privacy – ACISP 2005*, volume 3574 of *LNCS*, pages 267–279. Springer, 2005.

[97] J. S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354–1359, 1988.

[98] J. Lu, J. Kim, N. Keller, and O. Dunkelman. Differential and rectangle attacks on reduced-round SHACAL-1. In *Progress in Cryptology - INDOCRYPT 2006*, volume 4329 of *LNCS*, pages 17–31. Springer, 2006.

[99] M. G. Luby. *Pseudorandomness and cryptographic applications*. Princeton computer science notes. Princeton University Press, 1996.

[100] S. Lucks. A failure-friendly designprinciple forhashfunctions. In *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 474–494. Springer, 2005.

[101] S. Lucks and M. Daum. The story of alice and her boss: Hash functions and the blind passenger attack. Presented at the rump session of EUROCRYPT'05, May 2005. http://www.cits.rub.de/MD5Collisions/.

[102] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.

[103] K. Matusiewicz, T. Peyrin, O. Billet, S. Contini, and J. Pieprzyk. Cryptanalysis of FORK-256. In *Fast Software Encryption – FSE'07*, volume 4593 of *LNCS*, pages 19–38. Springer, 2007.

[104] K. Matusiewicz and J. Pieprzyk. Finding good differential patterns for attacks on SHA-1. In *Coding and Cryptography, WCC'2005*, volume 3969 of *LNCS*, pages 164–177. Springer, 2005.

[105] K. Matusiewicz, J. Pieprzyk, N. Pramstaller, C. Rechberger, and V. Rijmen. Analysis of simplified variants of SHA-256. In *Western European Workshop on Research in Cryptology – WEWoRC 2005*, volume P-74 of *Lecture Notes in Informatics*. Gesellschaft für Informatik, 2005.

[106] F. Mendel. Colliding message pair for 53-step HAS-160. Cryptology ePrint Archive, Report 2006/334, 2006. http://eprint.iacr.org/2006/334.

[107] F. Mendel, J. Lano, and B. Preneel. Cryptanalysis of reduced variants of the FORK-256 hash function. In *Topics in Cryptology – CT-RSA 2007*, volume 4377 of *LNCS*, pages 85–100. Springer, 2007.

[108] F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen. Analysis of step-reduced SHA-256. In *Fast Software Encryption – FSE 2006*, volume 4047 of *LNCS*, pages 126–143. Springer, 2006.

[109] F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen. The impact of carries on the complexity of collision attacks on SHA-1. In *Fast Software Encryption – FSE'06*, volume 4047 of *LNCS*, pages 278–292. Springer, March 2006.

[110] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001. available online from `http://www.cacr.math.uwaterloo.ca/hac/`.

[111] R. C. Merkle. *Secrecy, Authentication, and Public Key Systems*. PhD thesis, Stanford University, June 1979.

[112] R. C. Merkle. One way hash functions and DES. In G. Brassard, editor, *Advances in Cryptology - CRYPTO '89*, volume 435 of *LNCS*, pages 428 – 446. Springer-Verlag, 1989.

[113] I. Mironov and L. Zhang. Applications of SAT solvers to cryptanalysis of hash functions. In *Theory and Applications of Satisfiability Testing – SAT 2006*, volume 4121 of *LNCS*, pages 102–115. Springer, 2006.

[114] Y. Naito, Y. Sasaki, N. Kunihiro, and K. Ohta. Improved collision attack on MD4 with probability almost 1. In *Information Security and Cryptology – ICISC 2005*, volume 3935 of *LNCS*, pages 129–145. Springer, 2005.

[115] National Institute of Standards and Technology. Secure hash standard (SHS). FIPS 180-1, April 1995. Replaced by [116].

[116] National Institute of Standards and Technology. Secure hash standard (SHS). FIPS 180-2, August 2002.

[117] National Institute of Standards and Technology. Announcing the development of new hash algorithm(s) for the revision of federal information processing standard (FIPS) 180-2, secure hash standard. Federal Register, Vol. 72, No. 14, January 23 2007.

[118] K. Nishimura and M. Sibuya. Probability to meet in the middle. *Journal of Cryptology*, 2(1):13–22, February 1990.

[119] N. K. Park, J. H. Hwang, and P. J. Lee. HAS-V: A new hash function with variable output length. In D. R. Stinson and S. Tavares, editors, *Selected Areas in Cryptography – SAC 2000*, volume 2012 of *LNCS*, pages 202–216. Springer-Verlag, 2000.

[120] S. Park, S. H. Sung, S. Chee, and J. Lim. On the security of reduced versions of 3-pass HAVAL. In *Information Security and Privacy: 7th Australasian*

Conference, ACISP 2002, volume 2384 of *LNCS*, pages 267–270. Springer, 2002.

[121] J. Pieprzyk and D. Pointcheval. Parallel authentication and public-key encryption. In *Information Security and Privacy – ACISP 2003*, volume 2727 of *LNCS*, pages 387–401. Springer, 2003.

[122] N. Pramstaller, C. Rechberger, and V. Rijmen. Exploiting coding theory for collision attacks on SHA-1. In *Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings to appear*, volume 3796 of *LNCS*, pages 78–95. Springer, Dec 2005.

[123] B. Preneel. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, Katholieke Universiteit Leuven, 1993. Available from `http://homes.esat.kuleuven.be/~preneel/`.

[124] B. Preneel. The state of cryptographic hash functions. In *Lectures on Data Security*, volume 1561 of *LNCS*, pages 158–182. Springer-Verlag, 1999.

[125] B. Preneel, A. Bosselaers, and H. Dobbertin. RIPEMD-160: A strenghtened version of RIPEMD. In D. Gollmann, editor, *Fast Software Encryption – FSE '96*, volume 1039 of *LNCS*, pages 71–82. Springer-Verlag, 1997.

[126] B. Preneel, R. Govaerts, and J. Vandewalle. Hash functions based on block ciphers: A synthetic approach. In *Advances in Cryptology – CRYPTO '93*, volume 773 of *LNCS*, pages 368–378. Springer, 1993.

[127] J.-J. Quisquater and J.-P. Delescaille. How easy is collision search? application to DES (extended summary). In J.-J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology - EUROCRYPT'89*, volume 434 of *LNCS*, pages 429–434. Springer-Verlag, 1989.

[128] J.-J. Quisquater and J.-P. Delescaille. How easy is collision search. new results and applications to DES (abstract and results). In *Advances in Cryptology – CRYPTO '89*, volume 435 of *LNCS*, pages 408–413. Springer, 1989.

[129] V. Rijmen and E. Oswald. Update on SHA-1. In A. Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *LNCS*, pages 58–71. Springer-Verlag, Feb 2005.

[130] R. L. Rivest. The MD4 message digest algorithm. In A. J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90*, volume 537 of *LNCS*, pages 303–311. Springer-Verlag, 1991.

[131] R. L. Rivest. The MD5 message digest algorithm. Request for Comments (RFC) 1321, Internet Engineering Task Force, April 1992.

[132] P. Rogaway. Formalizing human ignorance. In *Progress in Cryptology – VIETCRYPT 2006*, volume 4341 of *LNCS*, pages 211–228. Springer, 2006.

[133] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance and collision resistance. In *Fast Software Encryption – FSE'04*, volume 3017, pages 371–388. Springer, 2004.

[134] M.-J. O. Saarinen. Cryptanalysis of block ciphers based on SHA-1 and MD5. In *Fast Software Encryption – FSE '03*, volume 2887 of *LNCS*, pages 36–44. Springer, 2003.

[135] Y. Sasaki, L. Wang, K. Ohta, and N. Kunihiro. New message difference for MD4. In *Fast Software Encryption – FSE '07*, volume 4593 of *LNCS*. Springer, 2007. to appear.

[136] M. Schläffer and E. Oswald. Searching for differential paths in MD4. In *Fast Software Encryption – FSE '06*, volume 4047 of *LNCS*, pages 242–261. Springer, 2006.

[137] B. Schneier. Nist hash workshop liveblogging (5). Schneier on Security Internet Blog, November 01, 2005. Available from `http://www.schneier.com/blog/archives/2005/11/nist_hash_works_4.html`.

[138] B. Schneier and J. Kesley. Unbalanced Feistel networks and block cipher design. In D. Gollmann, editor, *Fast Software Encryption – FSE'96*, volume 1039 of *LNCS*, pages 121–144. Springer-Verlag, 1996.

[139] A. N. Shiryayev. *Probability*, volume 95 of *Graduate Texts in Mathematics*. Springer-Verlag, 1984.

[140] M. Stevens. Fast collision attack on MD5. Cryptology ePrint Archive, Report 2006/104, 2006. `http://eprint.iacr.org/2006/104`.

[141] M. Stevens, A. Lenstra, and B. de Weger. Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In *Advances in Cryptology – EUROCRYPT'07*, volume 4515. Springer, 2007.

[142] D. R. Stinson. *Cryptography. Theory and practice*. Chapman & Hall/CRC, 2006.

[143] M. Sugita, M. Kawazoe, and H. Imai. Gröbner basis based cryptanalysis of sha-1. In *Fast Software Encryption – FSE 2007*, volume 4593 of *LNCS*. Springer, 2007.

[144] M. Szydlo and Y. Yin. Collision-resistant usage of MD5 and SHA-1 via message preprocessing. In *Topics in Cryptology – CT-RSA 2006*, volume 3860 of *LNCS*, pages 99–114. Springer, 2006.

[145] Telecommunications Technology Association of Korea. Hash function standard – part 2: Hash function algorithm standard (HAS-160). TTA Standard TTAS.KO-12.0011, 1998. `http://www.tta.or.kr/English/new/standardization/eng_ttastddesc.jsp?stdno=TTAS.KO-12.0011`.

[146] P. C. van Oorschot and M. J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, March 1999.

[147] B. van Rompay. *Analysis and design of cryptographic hash functions, MAC Algorithms and Block Ciphers*. PhD thesis, Katholieke Universiteit Leuven, 2004.

[148] B. van Rompay, A. Biryukov, B. Preneel, and J. Vandewalle. Cryptanalysis of 3-pass HAVAL. In *Advances in Cryptology – ASIACRYPT 2003*, volume 2894 of *LNCS*, pages 228–245. Springer, 2003.

[149] A. Vardy. The intractability of computing the minimum distance of a code. *IEEE Transactions on Information Theory*, 43(6):1757–1766, 1997.

[150] S. Vaudenay. On the need for multipermutations: Cryptanalysis of MD4 and SAFER. In *Fast Software Encryption – FSE 1994*, volume 1008 of *LNCS*, pages 286–297. Springer-Verlag, 1995.

[151] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the hash functions MD4 and RIPEMD. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT'05*, volume 3494 of *LNCS*, pages 1–18. Springer-Verlag, 2005.

[152] X. Wang, X. Lai, D. Feng, and H. Yu. Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199, August 2004. `http://eprint.iacr.org/`.

[153] X. Wang, X. Lai, D. Feng, and H. Yu. Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. Short talk presented at CRYPTO'04 Rump Session, 2004.

[154] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In *Advances in Cryptology – CRYPTO'05*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.

[155] X. Wang and H. Yu. How to break MD5 and other hash functions. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT'05*, volume 3494 of *LNCS*, pages 19–35. Springer-Verlag, 2005.

[156] X. Wang, H. Yu, and Y. L. Yin. Efficient collision search attacks on SHA-0. In *Advances in Cryptology – CRYPTO'05*, volume 3621, pages 1–16. Springer, 2005.

[157] Y. Yajima, Y. Sasaki, Y. Naito, T. Iwasaki, T. Shimoyama, N. Kunihiro, and K. Ohta. A new strategy for finding a differential path of SHA-1. In *Information Security and Privacy – ACISP 2007*, volume 4586, pages 45–58. Springer, 2007.

[158] H. Yoshida, A. Biryukov, C. De Canniére, J. Lano, and B. Preneel. Non-randomness of the full 4 and 5-pass HAVAL. In *Security in Communication Networks – SCN 2004*, volume 3352 of *LNCS*, pages 324–336. Springer-Verlag, 2005.

[159] H. Yoshida and A. Biryukow. Analysis of a SHA-256 variant. In *Selected Areas in Cryptography – SAC'05*, volume 3897 of *LNCS*, pages 245–260. Springer, 2006.

[160] H. Yu. *Cryptanalysis of Hash Functions and HMAC/NMAC*. PhD thesis, Shandong University, April 2007.

[161] H. Yu, X. Wang, A. Yun, and S. Park. Cryptanalysis of the full HAVAL with 4 and 5 passes. In *Fast Software Encryption – FSE '06*, volume 4047 of *LNCS*, pages 89–110. Springer, 2006.

[162] A. Yun, S. Sung, S. Park, D. Chang, S. Hong, and H.-S. Cho. Finding collision on 45-step HAS-160. In *Information Security and Cryptology – ICISC 2005*, volume 3935 of *LNCS*, pages 146–155. Springer, 2005.

[163] G. Yuval. How to swindle Rabin. *Cryptologia*, 3:187–189, 1979.

[164] Y. Zheng, J. Pieprzyk, and J. Seberry. HAVAL – a one-way hashing algorithm with variable length of output. In J. Seberry and Y. Zheng, editors, *Advances in Cryptology – AUSCRYPT'92*, volume 718 of *LNCS*, pages 83–104. Springer-Verlag, 1993.

Index