Part I
Introduction

Pour the initial value in a big cauldron and place it over a nice fire. Now slowly add salt if desired and stir well. Marinade your input bit string by appending some strengthened padding. Now chop the resulting bit string into nice small pieces of the same size and stretch each piece to at least four times its original length. Slowly add each single piece while continually stirring at the speed given by the rotation constants and spicing it up with some addition constants. When the hash stew is ready, extract a nice portion of at least 128 bits and present this hash value on a warm plate with some garnish.

Recipe 1: *Old Hash Recipe*

1 Hash functions

Contents

1.1 Authenticity and confidentiality

Authenticity and confidentiality have been big concerns in communications for millennia. One of the first known examples of confidentially sending a message goes back to around 440 BC to "The Histories" written down by Herodotus. According to Herodotus, Histiaeus sent an important message to his son-in-law Aristagoras. As this message was very sensitive he tattooed it on the shaved head of his most-trusted slave. Later, when enough hair had grown back, he sent his slave to Aristagoras with instructions to shave the head of the slave again.

There is a risk with hidden messages that somebody finds the hidden message and exposes it to unfriendly parties. This is especially the case for regular communications. Eventually the idea arose to encrypt messages so that they cannot be read even if intercepted and this has led to the development of *cryptology*, historically known as *the art of building and breaking ciphers.* Ciphers allow parties to securely communicate by encrypting their messages with some secret knowledge (the secret key) into unreadable ciphertext that can only be read by parties that possess the same secret knowledge. This form of encryption using a single secret key known to both sender and receiver is called *symmetric encryption.* The most famous ancient cipher, the Caesar cipher, is named after Julius Caesar who encrypted his most important messages with a secret number N by cyclically substituting each letter with the N-th next letter in the alphabet (if $N = 3$ then A goes to D, B to E, etc.).

Authentication of messages, the act of confirming that a message really has been sent by a certain party, usually was achieved by a combination of inspecting the message (e.g., verify its signature), the 'envelope' (e.g., is the wax seal intact) and/or

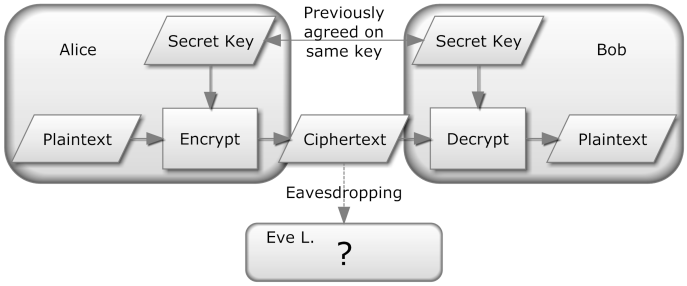

Figure 1: *Symmetric Encryption*

the messenger (e.g., is the messenger known and/or in service of either party). Symmetric encryption also provides some form of authentication. After all, no one else knows the secret key and is able to encrypt messages with it. However, this does not prevent the encrypted message from being purposely changed or simply repeated at an opportune moment by unfriendly parties. In general, authentication of someone (or the sender/creator of something) is achieved through mutual knowledge (such as a secret password), possession of a physical token (such as the king's seal) and/or distinguished marks (such as a known birthmark).

1.2 Rise of a digital world

With the invention of electronics, a new world was born. Digital information can be very cheaply stored and copied without loss of information. The rapid development of computers opened up a huge range of new possibilities in processing information. The explosive growth of telecommunication networks has made it very easy to quickly send information over vast distances to anywhere in the world. So not surprisingly, our society has become more and more dependent on information technologies in just a few decades.

The main advantage of digital information is that there is no longer a link between the information and its carrier. It can be copied endlessly without any loss of quality. The same information can be stored on, e.g., a hard disk, USB disk, mobile phone or a DVD, or sent through an Ethernet, Wi-Fi, GSM or satellite link.

This also introduces a large vulnerability as now authentication of information cannot be guaranteed anymore by simply looking at the physical carrier. Also, ciphers require two parties to first agree on a secret key. Given the vast number of possible parties in the world anyone might want to communicate privately with, establishing and storing secret keys for every possible connection between two parties becomes prohibitive.

Solving these security problems and many more that arose has expanded the field of cryptography. Modern cryptography has grown into a science that encompasses much more than only ciphers and has strong links with mathematics, computer science and (quantum) physics. One of the key innovations of modern cryptography that

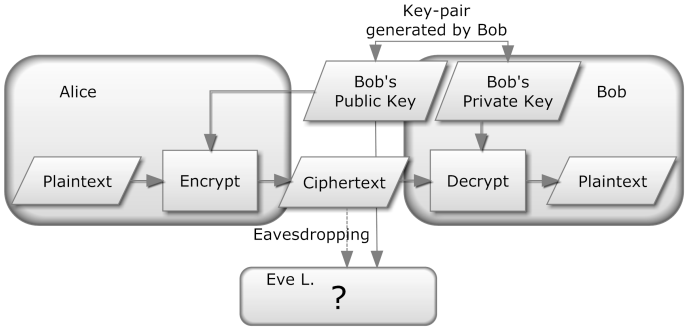

Figure 2: *Asymmetric or Public Key Encryption*

gave rise to new ideas and approaches is *public key encryption* (also called *asymmetric encryption*) [DH76, Mer78, RSA78]. Instead of using one secret key for both encryption and decryption, public key encryption uses a key pair consisting of a public key for encryption and a private key for decryption.

Suppose everyone publishes the public key part of their properly generated key pair in a public key listing (like a phone book). Now, securely sending a message to someone is possible without first exchanging a secret key in advance anymore, as you simply encrypt your message with his public key as found in the listing. Only he can now decrypt your ciphertext using the private key known only to him. This is a remarkable invention, although it depends heavily on the fact that no one can derive the private key from the public key. Furthermore, it requires a guarantee from the public key listing that every listed public key is generated by the associated person and not by an impostor.

Ideas similar to that of public key encryption have also led to *digital signatures* [RSA78]. With digital signatures you have a key pair as well which now consists of a signing key and a verification key. The verification key is made public and should be collected into a public listing such as that for public encryption keys. The signing key remains private and can be used to generate a digital signature for a message, which is then simply appended to the message. Anyone can verify the correctness of the digital signature of a message using the publicly listed verification key and thereby obtains proof of authenticity. However, only someone who knows the signing key can actually generate signatures. Similar to public key encryption, the security of digital signatures depends heavily on the fact that no one can derive the signing key from the verification key and also requires a guarantee from the public key listing that every listed verification key is generated by the associated person and not by an impostor.

1.3 Security frameworks

To analyze the security of cryptographic systems it is important to precisely define the desired notion of security. First, it needs to be clear what a possible adversary can and cannot do, e.g., whether he can eavesdrop communications or whether he is

able to alter messages. Then security notions can be expressed as problems for an adversary, such as finding a private key belonging to a given public key, or decrypting a message without knowledge of the private key, that are hopefully then shown to be hard to solve. However, there is no single good definition of how hard a problem is.

Most of modern cryptography uses one of the following three approaches to analyze the hardness of a problem: the information theoretic approach, the complexity theoretic approach and the system based approach. These approaches differ mainly on their assumptions about the computational limitations of an adversary. The notion of security becomes stronger when fewer limitations are assumed on the adversary. However, most public key cryptography requires at least some reasonable assumption about the limitations on the adversary, i.e., without any assumptions it is even possible that no secure solutions can be found. Each of these three approaches has its advantages and its practical use.

1.3.1 Information theoretic approach

The information theoretic approach makes no assumptions on the computational power of the adversary. This approach has been developed by Claude Shannon [Sha48] to find the fundamental limits on information processing operations such as compression, storage and communication. Shannon introduced the notion of entropy that quantifies information and which intuitively can be seen as the amount of uncertainty an entity has about a piece of information. More practically it can also be seen as the average number of bits you need to store the information.

This approach allows for unconditional security; security which is independent of the computing power of an adversary. It has been shown by Shannon [Sha49] that unconditional (or perfect) privacy protection can be achieved when the length of the uniformly randomly chosen encryption key is at least the length of the plaintext and the encryption key is only used once. More precisely, unconditional privacy requires that the entropy of the key is at least the entropy of the plaintext. Furthermore, unconditional message authentication can also be achieved at the cost of a large secret key.

The main advantage of this approach is that it allows unconditional security. However, the large secret keys which it usually requires makes it rather unpractical.

1.3.2 Complexity theoretic approach

Complexity theory tries to classify computational problems by their inherent difficulty. To quantify this difficulty, this approach utilizes a model of computation such as a Turing machine. In this model, algorithms are measured by the amount of time and/or the amount of space (memory) they use as a function in the size of the input. In general, algorithms are called *efficient* if their runtime and space are bounded by a polynomial function in the size of the input. An adversary is limited in this approach to a certain class of algorithms which is usually the class of efficient algorithms.

Computational problems are seen as an infinite collection of instances with a solution for every instance. To each such computational problem belongs a collection of

algorithms that can solve each instance of the problem (sometimes a certain probability of failure is allowed). As an adversary is limited to the use of efficient algorithms, computational problems that are generally assumed to have no efficient algorithms are of special interest. Examples are the *factoring problem* – finding the prime factors of a composite integer number – and the *discrete logarithm problem* – finding a number x such that $h = g^x \bmod p$ for a given prime p and $g, h \in \{1, \ldots, p-1\}$ [1]. In fact, the well-known RSA public-key encryption system [RSA78] would be broken if the factoring problem has an efficient algorithm. This approach allows the construction of cryptographic primitives (such as public key encryption, digital signatures, zero-knowledge proofs, etc.) for which it can be proven that breaking the cryptographic primitive implies the existence of an efficient algorithm that solves the underlying computational problem. Such primitives are often (a bit misleadingly) called *provably secure* although such proofs only reduce the primitive's security to the *assumed* hardness of some computational problem.

The advantages of this approach are that one can obtain provable security (based on a number of assumptions) and that secret keys can be smaller than in the information theoretic approach. It also has some inherent disadvantages. The distinction of an algorithm being efficient is mainly asymptotic. For concrete input lengths an algorithm with exponential runtime can be a lot faster than an algorithm with polynomial runtime, e.g., due to a large difference in the constant factor. This implies that provable security gives no information on the security of concrete instances. Also, for real world instances an efficient algorithm, such as an encryption algorithm, may have an impractically large runtime; making it impractical for use in cryptographic systems.

1.3.3 System based approach

The system based approach is based on the real world efficiency of software and hardware implementations and tries to produce very practical cryptographic primitives. Analyzing the hardness of a problem instance is based on estimating the necessary real-world resources (in computing power, memory, dedicated hardware, etc) as required by the best known algorithm that solves the problem. Compared to the complexity theoretic approach wherein an algorithm is deemed efficient if its asymptotic complexity is bounded by a polynomial function, the system based approach analyzes concrete problem instances and deems solving a concrete problem instance as feasible if the necessary real-world resources can be obtained within a certain reasonable monetary budget.

Concrete cryptographic primitives are designed to be fast and to make breaking them a hard problem. Similar to a cat-and-mouse game, several attack principles have been invented over the years to analyze and break older primitives, each leading to new designs that try to avoid practical attacks based on those principles.

The advantage of this approach is that it results in very fast cryptographic primitives. The main disadvantages are that the security is mainly based on preventing

1. $g^x \bmod p$ denotes the remainder of g^x after division by p

known attacks and that the designs might seem ad-hoc. The remainder of this thesis mainly uses the system based approach.

1.4 Cryptographic hash functions

1.4.1 One-way functions

In computer science and cryptography there is a type of function that is both theoretically and practically important: *one-way functions*. A one-way function maps a huge (possibly infinite) domain (e.g., of all possible messages) to some (possibly infinite) range (e.g., all bit strings of length 256). The predicate one-way means that such a function is easy to compute whereas it is hard to invert, i.e., for a given output it is hard to find an input that maps to that output. The one-wayness property is not rigorously defined and depends on the security framework used to analyze the hardness of the inverting problem. As the very definition of a one-way function depends on the security framework used, so does the question of their existence.

The information theoretic view leads to the most negative result: one-way functions do not even exist in this view. An adversary with unlimited computing power can simply test every possible input until he finds one that results in the given output.

All known one-way functions in the complexity theoretical approach are actually based on (well-established) assumptions. The existence of one-way functions is thus assumed. Nevertheless, finding an actual proof remains a very interesting problem. In fact, a proof of their existence would also prove that the complexity classes P and NP are distinct which thereby resolves one of the foremost unsolved questions of computer science. The existence of one-way functions also implies the existence of various cryptographic primitives such as pseudo-random generators [ILL89], digital signature schemes [Rom90], zero-knowledge protocols, message authentication codes and commitment schemes.

In cryptography there are several problems that are *assumed* to be hard and can be used to define a one-way function. The earlier mentioned factoring problem leads to the one-way function that maps two very large (say at least 2048-bits) prime numbers p and q to their product $p \cdot q$. Similarly, the discrete logarithm problem leads to the one-way function that maps a number e to $g^e \bmod p$, for a given integer g and a large prime number p.

For cryptography, an important class of one-way functions is the class of one-way *hash* functions, or simply *hash functions*, that operate on bit strings and map bit strings of arbitrary length to a bit string of a fixed length (such as 256 bits) which is called the *hash*.

1.4.2 Cryptographic requirements of hash functions

A major application for hash functions is found in constructing efficient digital signature schemes. In 1978, Rabin [Rab78] introduced the idea of signing the hash of

a document M instead of directly signing it[2]. Signing a large message directly with a public-key crypto system is slow and leads to a signature as large as the message. Reducing a large message to a small hash using a hash function and signing the hash is a lot faster and leads to small fixed-size signatures. Forging a signature then requires either breaking the public-key crypto system or finding a document M' that has the same hash as a different signed document M.

In this case, the security of digital signatures is also based on the hardness of the problem of finding a *collision*, i.e., finding two different documents M and M' that have the same hash. Since if either M or M' is signed, then the corresponding signature is also valid for the other document, resulting in a successful forgery. The actual value of the hash is unimportant as long as both documents have the same hash.

Here it already becomes clear that for cryptographic purposes the one-wayness of a hash function is not enough. For cryptographic use, the following nine cryptographic properties of finite families $\mathcal{F}$ of hash functions with identical output bit length are considered: Pre, aPre, ePre, Sec, aSec, eSec, Coll, MAC and PRF. For a more thorough treatment of these properties, we refer to [RS04], [BR06a] and [BR07].

Pre, ePre and aPre: The first three are based on variations of the problem of finding a pre-image M for a given hash h and are definitions of one-wayness. Pre, which stands for *pre-image resistance*, requires that given a uniformly randomly chosen hash function f from a family $\mathcal{F}$ and a uniformly randomly chosen hash h it is hard to find a pre-image $M \in f^{-1}(h)$. The stronger notion ePre (*everywhere pre-image resistance*) allows the adversary to choose the hash h before f is uniformly randomly chosen from $\mathcal{F}$. The third notion aPre (*always pre-image resistance*) allows the adversary to choose the hash function f before the hash h is uniformly randomly chosen and given to the adversary.[3]

Sec, eSec and aSec: The next three are based on variations of the problem of finding another message M' that has the same hash as a given message M. Sec (*second pre-image resistance*) requires f and M to be uniformly randomly chosen. The stronger eSec (*everywhere Sec*) allows the adversary to choose M before f is uniformly randomly chosen and aSec (*always Sec*) allows the adversary to choose f before M is uniformly randomly chosen and given to the adversary.[3]

Coll: Coll (*collision resistance*) is the property that finding two different messages M and M' that have the same hash $f(M) = f(M')$, where f is a hash function chosen randomly from $\mathcal{F}$ is a hard problem. This property is often the first to be broken and hence requires special attention.

2. Although at the time no such hash functions were available. The first design towards hash functions as used today is MD4 which was introduced in 1990.
3. These variations of Pre (and Sec) may seem to be very similar; nevertheless, their distinctions are theoretically important. For instance in the case of a fixed hash function, where there is no hash function family to speak of, aPre and aSec are the only applicable notions.

MAC: The MAC (*message authentication code unforgeability*) property views the entire hash function family $\mathcal{F}$ as a single keyed hash function f_K where K is a randomly chosen secret key. The adversary does not get direct access to K and f_K, instead the adversary can make queries q_i to an oracle that responds with the output hashes $r_i = f_K(q_i)$. The MAC property requires that the problem of finding a message M and its correct hash under f_K, where M is not one of the queries q_i, is hard.

PRF: For the PRF (*pseudo random function*) property, the problem of distinguishing between an oracle to f_K for a randomly chosen secret K and a random function oracle that responds to queries q_i with randomly chosen hashes r_i must be hard.

In the case of a fixed hash function f instead of a hash function family $\mathcal{F}$, only three properties are considered: pre-image resistance, second pre-image resistance and collision resistance. The pre-image resistance and second pre-image resistance properties are identical to the above aPre and aSec notions, respectively.

However, when formally defining collision resistance for such a fixed hash function f one runs into the *foundations-of-hashing dilemma* [Rog06]. Collision resistance is an intuitive concept and one would like to mathematically define collision resistance as the hardness of some problem or equivalently as some statement "*there is no efficient algorithm that produces collisions*". Unfortunately, for every collision $M \neq M'$ of f, there is a trivial algorithm that simply outputs that collision: M and M'. Nonetheless, if there are no collisions known then one cannot actually write down such a trivial algorithm. Thus one would like to think of collision resistance as the statement "*there is no* known *efficient algorithm that produces collisions*", which may be impossible to define mathematically. Rogaway [Rog06] treats the foundations-of-hashing dilemma in detail and provides a formal way to handle collision resistance, but does not provide a clear definition for collision resistance.

Thus in this thesis we do not define the collision resistance property for a fixed hash function in terms of some mathematical problem that must be hard, rather we view collision resistance as the property that there are no known explicit efficient algorithms. This informal definition of collision resistance is sufficient for the remainder of this thesis as we focus on counter-examples to the collision resistance of fixed hash functions.

1.4.3 General attacks

Even though we would prefer breaking these security properties to be impossible, often there exists a general attack breaking a security property that thereby presents a fundamental upper bound for an adversary's attack complexity. A hash function is called *broken* when there exists a known explicit attack that is faster than the general attack for a security property. It must be noted that even unbroken hash functions may be insecure in the real-world, e.g., a general attack becomes feasible due to a too small hash bit length compared to the possible real-world computational power of adversaries.

The best known general attack to break Pre, aPre, ePre, Sec, aSec and eSec is a brute force search, where hashes $f(M')$ are computed for randomly chosen messages M' until a message M' is found where $f(M')$ is the target hash value (and $M' \neq M$ for Sec, aSec, eSec). For a hash function (family) with an output hash size of N bits, this attack succeeds after approximately 2^N evaluations of the hash function. Already for $N \geq 100$ this attack is clearly infeasible in the real world for the present day and near future.

To find collisions, one can do a lot better with a general attack based on the birthday paradox. The birthday paradox is the counter-intuitive principle that for groups of as few as 23 persons there is already a chance of about one half of finding two persons with the same birthday (assuming all birthdays are equally likely and disregarding leap years). Compared to finding someone in this group with your birthday where you have 23 independent chances and thus a success probability of $\frac{23}{365} \approx 0.06$, this principle is based on the fact that there are $\frac{23*22}{2} = 253$ distinct pairs of persons. This leads to a success probability of about 0.5 (note that this does not equal $\frac{253}{365} \approx 0.7$ since these pairs are not independently distributed).

For a given hash function with output size of N bits, this general algorithm succeeds after approximately $\sqrt{\pi/2} \cdot 2^{N/2}$ evaluations of the hash function [vOW99]. This means that for a hash size of $N = 128$ bits (which was commonly used until recently) finding a collision needs approximately $2^{64.3} \approx 22 \cdot 10^{18}$ evaluations, which is currently just in reach for a large computing project[4], whereas inverting the hash function takes about a factor of $15 \cdot 10^{18}$ longer.

1.4.4 From compression function to hash function

With the need for secure practical hash function designs for use in digital signatures schemes well known [Rab78, Yuv79, DP80, Mer82, Rom90], the first attempts to construct a hash function were made in the 1980s. An immediately recognized design approach to tackle the problem of arbitrary length inputs was to base the security of the hash function on the security of a function with fixed size inputs, such as a block cipher which is an already well studied cryptographic primitive. More generally for the purpose of hash functions, such a fixed input size function is called a *compression function* as it must have an output length smaller than its input length.

The compression function is then repeatedly used in some mode of operation to process the entire message. The well known Merkle-Damgård construction (named after the authors of the independent papers [Mer89, Dam89] published in 1989) describes exactly how to construct a hash function based on a general compression function in an iterative structure as is depicted in Figure 3. Since these papers have proven that the hash function is collision resistant if the underlying compression function is collision resistant, the majority of currently used hash functions are based on

4. The distributed computing project MD5CRK started in March 2004 a brute force search for collisions for the hash function MD5. The project was stopped in August 2004 due to breakthrough cryptanalytic research [WFLY04, WY05]. As a rough estimate, the attack can currently be done in one year on 40.000 quad-core machines or on 1000 high-end graphics cards.

this Merkle-Damgård construction.

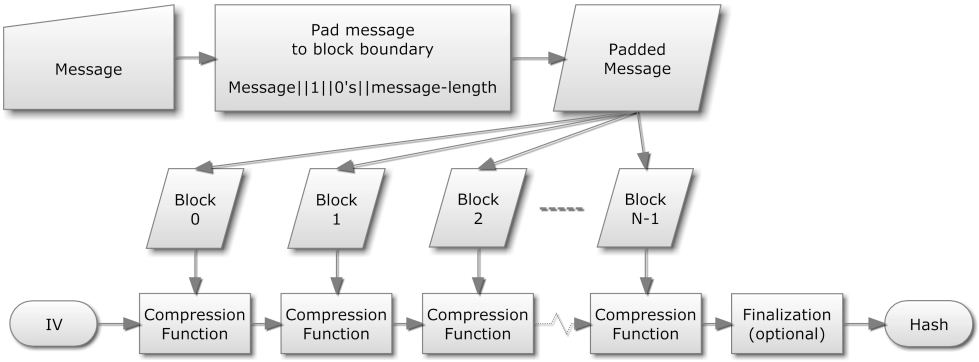

Figure 3: *Merkle-Damgård Construction*

The construction builds a hash function based on a compression function that takes two inputs: a chaining value or IHV_{in} (*Intermediate Hash Value*) of K bits and a message block of N bits, and outputs a new K-bit IHV_{out}. An input message is first padded with a single '1'-bit followed by a number X of '0'-bits and lastly the original message length encoded into 64 bits. The number X of '0'-bits to be added is defined as the lowest possible number so that the entire padded message bit length is an integer multiple of the message block length N. The padded message is now split into blocks of size exactly N bits. The hash function starts with a fixed public value for IHV_0 called the IV (*Initial Value*). For each subsequent message block it calls the compression function with the current IHV_i and the message block M_i and store the output as the new IHV_{i+1}. When we focus only on the compression function, we write IHV_{in} and IHV_{out} for the input IHV_i and the output IHV_{i+1}, respectively. After all blocks are processed it outputs the last IHV_N after an optional finalization transform.

The Merkle-Damgård construction has several weaknesses, none of which pose a real-world threat against the use of currently commonly used hash functions based on the construction. In 2004, Joux [Jou04] showed that finding a *multi-collision* – defined as 2^t messages that all have the same hash value – costs only about t times a single collision attack. Furthermore, cascaded hash functions $f(M) = g(M)||h(M)$, obtained by concatenating the output from two different hash functions, were often expected to yield a more secure hash function than either g and h are. However, if either g or h is based on Merkle-Damgård then using multi-collisions Joux showed that f is as secure as the most secure hash function of g and h.

In 2005, a general second pre-image attack against Merkle-Damgård based hash functions was published [KS05] which can find a second pre-image for a given message consisting of about 2^k blocks in about $2^{n-k+1} + k2^{n/2+1}$ evaluations of the hash function instead of the 2^n evaluations of the brute force attack.

In 2006, it has been shown that the Merkle-Damgård construction preserves also

the ePre property from a compression function besides the Coll property, but fails to preserve Pre, aPre, Sec, aSec and eSec ([ANPS07, BR06a]). Several new constructions have been proposed that preserve many (but not all) of these properties, e.g., ROX [ANPS07], EMD [BR06a] and ESh [BR07].

1.4.5 The MD4 style family of hash functions

Based on the work by Merkle and Damgård and 10 years after the idea of using hash functions in digital signatures [DP80] was introduced, Ron Rivest presented in 1990 the dedicated hash function MD4 [Riv90a, Riv90b] as a first attempt using the Merkle-Damgård construction. MD4 was quickly superseded by MD5 [Riv92] in 1992 due to security concerns [dBB91]. MD5 has found widespread use and remains commonly used worldwide.

MD4 and MD5 have formed an inspiration for other hash function designs that have followed over the years. Several of them can be seen as part of a MD4 style family of hash functions as they are based on Merkle-Damgård and have the same basic design of a compression function:

- SHA-0 [NIS93] (designed by the NSA, the National Security Agency of the USA, in 1993);
- SHA-1 [NIS95] (the replacement of SHA-0 by the NSA in 1995 after undisclosed security concerns were found in SHA-0);
- SHA-2 [NIS02, NIS08, NIS11] consisting of SHA-224, SHA-256, SHA-384 and SHA-512 (designed in 2002 by NSA);
- RIPEMD [BP95];
- RIPEMD-160 [DBP96] consisting of strengthened versions of RIPEMD;
- and many others.

An MD4 style compression function (Figure 4) takes a message block and generates an expanded message block split into R pieces $m_0, \ldots, m_{R-1}$ whose total bit length is at least three times the block length (MD5 and SHA-1 use four and five times, respectively). It initializes a working state with the input IHV called IHV_{in} and iteratively updates this working state with a *step function* and consecutive pieces $m_0, \ldots, m_{R-1}$, very similar to the Merkle-Damgård construction. Lastly it outputs the new IHV called IHV_{out} as the sum of the input IHV_{in} and the last working state. The security of the compression function mainly depends on highly complex bit dependencies in the step function output and a high expansion factor (which is the expanded message block bit length divided by the input message block bit length).

All members of the MD4 style hash function family have different step functions; nevertheless, they also have many similarities. In the case of MD5 the step function is sketched in Figure 5, where the working state consists of four variables A, B, C

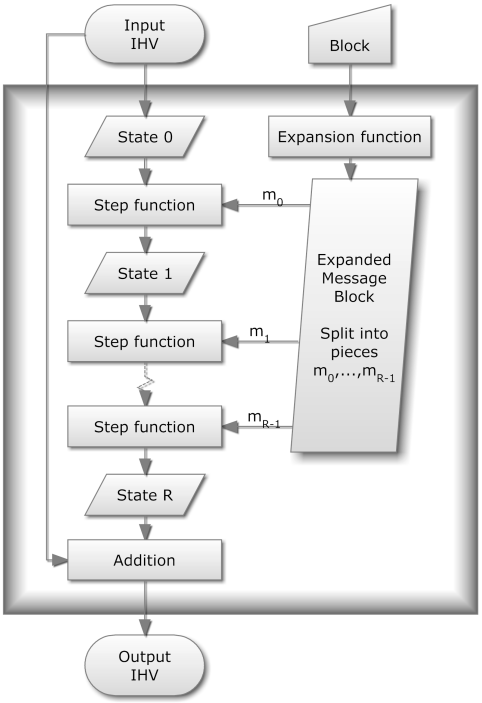

Figure 4: *MD4 Style Compression Function*

and D which are both seen as elements of $\mathbb{Z}_{2^{32}}$ and as 32-bit strings.[5] The variable A is updated by adding a message piece m_i, an addition constant K_i and the result $F(B, C, D)$ of a bit-wise function F (the i-th bit of the result only depends on the i-th bits of the inputs). Next, A is bit-wise left rotated by s (the rotation constant) bits and finally the input value of B is added. Let $\widehat{A}$ denote the updated variable A:

$$\widehat{A} = B + RL(A + m_i + K_i + F(B, C, D), s)$$

The output working state A', B', C' and D' consists of the rotated variables: $A' = D$, $B' = \widehat{A}$, $C' = B$ and $D' = C$. The step function itself varies a little between the R steps in MD5 in that the constant values K_i and s and the bit-wise function F are changed.

The step function is non-linear so that the compression function cannot be simply described as a linear system of equations that can easily be solved and thus inverted. Moreover, it tries to create very complex dependencies of IHV_{out} on all the message block bits. It does so by mixing different mathematical operations (like modular addition, bit-wise functions and bit-wise rotation) for which there is no unified 'calculus' that allows to simplify the mathematical expression defining the compression

5. Throughout this thesis we use $\mathbb{Z}_{2^{32}}$ as shorthand for $\mathbb{Z}/2^{32}\mathbb{Z}$.

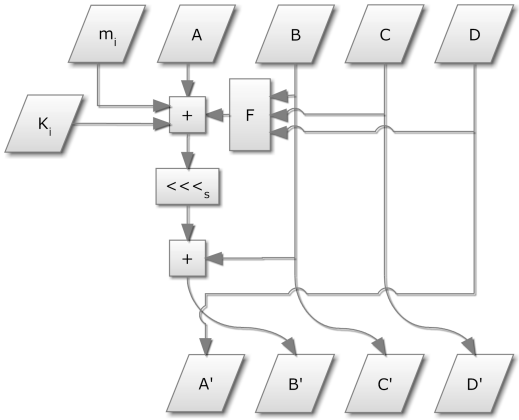

Figure 5: *Step Function of MD5's Compression Function*

function. This effect is amplified by using each message bit multiple times (4 times in MD5) over various spread-out steps. As a result of the lack of such a calculus, solving a set of equations over the compression function that results in a collision or pre-image attack appears to be very difficult.

Note that this step function, like the one of MD4 and other MD4-style hash functions, is efficiently and uniquely invertible given the output working state and the message piece. Therefore, the addition at the end of the compression function is necessary to avoid that inverting the entire compression function is easy. Without this addition, the inverting problem reduces to finding R pre-images of the step function, which can be done just as fast as evaluating the compression function.

1.5 Differential cryptanalysis

The most successful type of cryptographic analysis of the MD4 style hash function family is *differential cryptanalysis*. In differential cryptanalysis one looks at two evaluations of a given hash function at the same time. By analyzing how differences between those two evaluations caused by message differences propagate throughout the computation of hash function, one can try to control those differences and try to build an efficient attack. This type of analysis is mainly used for collision attacks, but sometimes extends to second pre-image attacks.

Differential cryptanalysis has been publicly known since the first published attacks against the DES (Data Encryption Standard) cipher by Biham and Shamir since 1990 [BS90, BS91, BS92]. The initial differential cryptanalysis by Biham and Shamir was based on the bitwise XOR difference and they apply this new technique to a number of ciphers and even a few cipher-based hash functions.

The technique was quickly generalized from the bitwise XOR difference to the modular difference (modulo 2^{32}) and applied to the MD4 style hash function family [Ber92]. Differential cryptanalysis using the modular difference was not successful,

since it was not possible to effectively deal with the bitwise operations (the boolean function and the bitwise rotation). Later, in 1995, Dobbertin [Dob96] was partially successful as he was able to find collisions for the compression function of MD5, but this did not extend to a collision attack against MD5 itself. In the case of MD4 he was more successful. Dobbertin [Dob98] was able to find collisions for MD4 using differential cryptanalysis modulo 2^{32} in 1996.[6]

Major cryptanalytic breakthroughs, based on a combination of the XOR difference and the modular difference, were found in the year 2004 when collisions were presented for SHA-0 by Biham et al. [BCJ+05] and MD4, MD5, HAVAL-128 and RIPEMD [WFLY04] by Xiaoyun Wang et al. These attacks have set off a new innovation impulse in hash function theory, cryptanalysis and practical attacks. The recent advances have resulted in a loss of confidence in the design principles underlying the MD4 style hash functions. In light of this, NIST has started a public competition in 2007 to develop a new cryptographic hash function SHA-3 [NIS07] to be the future de facto standard hash function. At the end of 2008, 51 out of 64 submitted candidate hash functions were selected for the first round of public review. Mid 2009, only 14 were selected to advance to the second round. After another year based on public feedback and internal reviews five SHA-3 finalists were selected at the end of 2010: BLAKE [AHMP10], Grøstl [GKM+11], JH [Wu11], Keccak [BDPA11] and Skein [FLS+10]. NIST will decide the winner in 2012 and name it SHA-3.

6. Hans Dobbertin fell ill suddenly in the late spring of 2005 and passed away on the 2nd of February 2006.