OULU 2012
C 433

ACTA

UNIVERSITATIS OULUENSIS

Kimmo Halunen

HASH FUNCTION SECURITY

CRYPTANALYSIS OF THE VERY SMOOTH HASH AND MULTICOLLISIONS IN GENERALISED ITERATED HASH FUNCTIONS

UNIVERSITY OF OULU GRADUATE SCHOOL;
UNIVERSITY OF OULU, FACULTY OF TECHNOLOGY,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING;
INFOTECH OULU

ACTA UNIVERSITATIS OULUENSIS
C Technica 433

KIMMO HALUNEN

HASH FUNCTION SECURITY

Cryptanalysis of the Very Smooth Hash and multicollisions in generalised iterated hash functions

Academic dissertation to be presented with the assent of the Doctoral Training Committee of Technology and Natural Sciences of the University of Oulu for public defence in Auditorium TS101, Linnanmaa, on 16 November 2012, at 9 a.m.

UNIVERSITY OF OULU, OULU 2012

Acta Univ. Oul. C 433, 2012

Supervised by
Professor Juha Röning

Reviewed by
Professor Bart Preneel
Doctor Martijn Stam

ISBN 978-951-42-9965-0 (Paperback)
ISBN 978-951-42-9966-7 (PDF)

ISSN 0355-3213 (Printed)
ISSN 1796-2226 (Online)

Cover Design
Raimo Ahonen

JUVENES PRINT
TAMPERE 2012

Halunen, Kimmo, Hash function security. Cryptanalysis of the Very Smooth Hash and multicollisions in generalised iterated hash functions

University of Oulu Graduate School; University of Oulu, Faculty of Technology, Department of Computer Science and Engineering; Infotech Oulu, P.O. Box 4500, FI-90014 University of Oulu, Finland

Acta Univ. Oul. C 433, 2012
Oulu, Finland

Abstract

In recent years, the amount of electronic communication has grown enormously. This has posed some new problems in information security. In particular, the methods in cryptography have been under much scrutiny. There are several basic primitives that modern cryptographic protocols utilise. One of these is hash functions, which are used to compute short hash values from messages of any length.

In this thesis, we study the security of hash functions from two different viewpoints. First of all, we analyse the security of the Very Smooth Hash against preimage attacks. We develop an improved method for finding preimages of Very Smooth Hash, compare this method with existing methods and demonstrate its efficiency with practical results. Furthermore, we generalise this method to the discrete logarithm variants of the Very Smooth Hash.

Secondly, we describe the methods for finding multicollisions in traditional iterated hash functions and give some extensions and improvements to these. We also outline a method for finding multicollisions for generalised iterated hash functions and discuss the implications of these findings. In addition, we generalise these multicollision finding methods to some graph-based hash functions.

Keywords: cryptography, hash functions, information security, multicollisions, Very Smooth Hash

Halunen, Kimmo, Hash-funktioiden turvallisuus. Very Smooth Hash -funktion analyysi sekä monitörmäykset yleistetyissä iteroiduissa hash-funktioissa
Oulun yliopiston tutkijakoulu; Oulun yliopisto, Teknillinen tiedekunta, Tietotekniikan osasto; Infotech Oulu, PL 4500, 90014 Oulun yliopisto
Acta Univ. Oul. C 433, 2012
Oulu

Tiivistelmä

Viime vuosina digitaaliseen tiedonsiirtoon perustuva tiedonsiirto on yleistynyt valtavasti. Tästä on seurannut monia uusia tietoturvaongelmia. Tässä yhteydessä erityisesti tiedon suojaamiseen käytetyt kryptografiset menetelmät ovat olleet tarkastelun kohteena. Hash-funktiot ovat yksi käytetyimmistä työkaluista nykyisissä kryptografisissa protokollissa.

Tässä väitöskirjassa tarkastellaan hash-funktioiden turvallisuutta kahden eri tutkimusongelman kautta. Aluksi tutkitaan Very Smooth Hash -funktion turvallisuutta alkukuvien löytämistä vastaan. Alkukuvien löytämiseksi esitetään parannettu menetelmä, jota arvioidaan teoreettisilla ja käytännöllisillä menetelmillä. Tämä parannettu menetelmä yleistetään koskemaan myös Very Smooth Hashin muunnoksia, jotka perustuvat diskreetin logaritmin ongelmaan.

Toisena tutkimuskohteena ovat iteroitujen hash-funktioiden yleistykset ja monitörmäykset. Aluksi esitellään perinteisiin iteroituihin hash-funktioihin liittyviä monitörmäysmenetelmiä. Tämän jälkeen tutkitaan iteroitujen hash-funktioiden yleistyksiä ja osoitetaan, että aiemmat monitörmäysmenetelmät voidaan laajentaa koskemaan myös näitä yleistyksiä. Lopuksi tutkitaan graafeihin perustuviin hash-funktioihin liittyviä monitörmäysmenetelmiä ja osoitetaan, että iteroitujen hash-funktioiden monitörmäysmenetelmä voidaan osittain yleistää koskemaan myös graafeihin perustuvia hash-funktioita.

Asiasanat: hash-funktiot, kryptografia, monitörmäykset, tietoturva, Very Smooth Hash

Acknowledgements

The journey to complete my Ph.D. thesis has been quite long and there are many people and organisations that I want to thank for helping me reach this goal. The research presented in this thesis has been carried out at the Oulu University Secure Programming Group (OUSPG) of the Department of Computer Science and Engineering, University of Oulu. For financial support, I thank Infotech Oulu, Tauno Tönning Research Foundation and Emil Aaltonen Foundation.

It would have been impossible to make it through this process without all the great people that I have had to help me and to work with. First of all, I would like to thank my advisor Professor Juha Röning for his guidance and support throughout the Ph.D. process. I would also like to thank Docent Juha Kortelainen, who has given great advice and mentoring to me and solved many of the common research problems that we tackled.

Special thanks go to the other co-authors of the papers included in this thesis, M.Sc. Tuomas Kortelainen and M.Sc. Pauli Rikula. It has been a privilege and a pleasure to work with such talented people. I also thank the reviewers of this thesis, Professor, Dr. Bart Preneel and Dr. Martijn Stam, for their valuable comments that helped to improve my thesis.

In addition, the great folks of OUSPG deserve all my thanks for helping me with all the practical aspects of information security and giving a fun and warm atmosphere to our workplace. I extend my greatest thanks to M.Sc.(Tech.) Teemu Tokola, who has been a true friend for over fifteen years and helped me find my way into OUSPG. Also all my friends both near and far have been a valuable source of advice, humour and encouragement. Thank you all for being my friends.

I also owe much gratitude towards my family; my siblings, parents, grandparents, aunts and uncles all deserve my thanks. Especially, my parents, Lea and Seppo, have always been supportive of my studies and academic career and have encouraged me to make most of my education. For that, I am very grateful for them.

Finally, I want to thank my son, Lenni, who has shown me that all learning and success come through persistence despite failures. Last and definitely most importantly, I want to thank my wife, Heidi, for all her love throughout the years and all the encouragement and comfort during the hard times and all the shared joy of my successes.

Abbreviations and notation

$\vec{0}$	*The zero vector, i.e. a vector with zero in all its coordinates*
$\lceil a \rceil$	*The ceiling function, i.e. the smallest integer greater than or equal to a*
$\lfloor a \rfloor$	*The floor function, i.e. the greatest integer less than or equal to a*
$\prec$	*A partial order (in some set), i.e. a transitive, antisymmetric and irreflexive relation*
$\prec_\alpha$	*The partial order in the set of symbols of the word α induced by the word α, i.e. $a \prec_\alpha b$ iff all occurrences of a in α precede the first occurrence of b in α*
$\lvert A \rvert$	*Cardinality of A, i.e. the number of elements in the set A*
$\lvert w \rvert$	*Length of w, i.e. the number of symbols in the word w*
$\lvert w \rvert_a$	*The number of occurrences of the symbol a in the word w*
A^+	*Kleene plus, i.e. the set of all possible words from the alphabet A with length ≥ 1*
A^*	*Kleene star, i.e. the set of all possible words from the alphabet A, including the empty word ε*
$\mathrm{alph}(w)$	*The set of symbols occurring in the word w, i.e. the set $\{a \in A \mid \lvert w \rvert_a > 0\}$*
ε	*The empty word, i.e. word with no symbols and of length 0*
$\mathbb{F}_q$	*A finite field of order q*
$\emptyset$	*The empty set, i.e. a set containing no elements; the set with the cardinality 0*
$\mathbb{N}$	*The set of all natural numbers, i.e. $\{0,1,2,\ldots\}$*
$\mathbb{N}_+$	*The set of all positive natural numbers, i.e. $\{1,2,\ldots\}$*
$O(f)$	*The big O of f, i.e. the set of functions asymptotically greater than f*
$\Omega(f)$	*The big Ω of f, i.e. the set of functions asymptotically smaller than f*
$\Theta(f)$	*The big Θ of f, i.e. the set of functions asymptotically equal to f*
$\mathbb{Z}$	*The set of integers, i.e. $\{\ldots,-2,-1,0,1,2,\ldots\}$*
$\mathbb{Z}_n$	*The set of integers modulo n, i.e. $\{0,1,2,\ldots,n-1\}$*
3C	*A method of constructing hash functions by adding an extra XOR-sum calculation in the iteration process*
AES	*Advanced Encryption Standard, a block cipher standardised by NIST in 2001*

ASCII	*American Standard Code for Information Interchange*
CEILIDH	*A torus-based public key cryptosystem*
DES	*Data Encryption Standard, a block cipher standardised by NIST in 1976*
DLP	*The discrete logarithm problem, see Definition 1 on page 29*
FIL-RO	*A fixed input length random oracle, i.e. a function that accepts messages of a fixed length as input and outputs a value from the range uniformly and at random*
FIPS 186-3	*Digital Signature Standard published by the National Institute of Standards and Technology*
GB	*Gigabyte, a unit of memory, i.e.* 10^9 *bytes*
GHz	*Gigahertz, a unit of frequency, i.e.* 10^9 *hertzs*
GOST	*Russian governmental standard, in this work, it refers to the hash function standard defined in GOST R 34.11-94 and GOST 34.311-95*
HAIFA	*Hash iterative framework, a method of constructing hash functions from compression functions*
HMAC	*Hash-based Message Authentication Code, a method of constructing message authentication codes by utilising a cryptographic hash function*
IEC	*International Electrotechnical Commission, a standards organisation*
ISO	*International Organization for Standardization*
MAC	*Message Authentication Code, a short piece of information for authenticating a message*
MD4	*Message Digest 4, a hash function*
MD5	*Message Digest 5, a hash function*
MDC-2	*Modification Detection Code 2, a hash function*
NFS	*Number field sieve, a factoring method*
NIST	*National Institute of Standards and Technology*
NMCAS	*Nested Multicollision Attack Schema, a method for finding multicollisions for generalised iterated hash functions*
NMSRVS	*Nontrivial Modular Square Root of Very Smooth numbers, a number theoretic problem, see Definition 5 on page 42*
QS	*Quadratic sieve, a factoring method*
ROX	*A method of constructing hash functions from compression functions, see Andreeva* et al. *(2007)*
RSA	*Rivest Shamir Adleman, an encryption method based on the hardness of the modular root problem*

SHA	*Secure Hash Algorithm, a family of hash functions*
VIL-RO	*A variable input length random oracle, i.e. a function that accepts messages of any length as input and outputs a value from the range uniformly and at random*
VSH	*Very Smooth Hash, a hash function*
VSH-DL	*Discrete logarithm variant of the Very Smooth Hash*
XOR	*The exclusive or -operation*
XTR	*A method for fast arithmetic in finite fields*

List of original articles

This thesis is based on the following publications that are referred to in the text by their Roman numerals (I–VII).

I Halunen K, Rikula P & Röning J (2008) On the Security of VSH in Password Schemes. Proceedings of the Third International Conference on Availability, Reliability and Security (ARES 2008): 828–833.

II Halunen K, Rikula P & Röning J (2009) Finding Preimages of Multiple Passwords Secured with VSH. Proceedings of the Fourth International Conference on Availability, Reliability and Security (ARES 2009): 499–503.

III Halunen K, Kortelainen J & Kortelainen T (2010), Combinatorial Multicollision Attacks and Generalized Iterated Hash Functions. Proceedings of Australasian Information Security Conference 2010 (AISC 2010): 86–93.

IV Kortelainen T, Kortelainen J & Halunen K (2010) Variants of Multicollision Attacks on Iterated Hash Functions. Proceedings of the Sixth China International Conference on Information Security and Cryptology (Inscrypt 2010): 139–154.

V Kortelainen J, Halunen K & Kortelainen T (2010) Multicollision Attacks and Generalized Iterated Hash Functions. Journal of Mathematical Cryptology 4(3): 239–270.

VI Halunen K & Röning J (2010) Preimage Attacks Against Variants of Very Smooth Hash. Proceedings of the Fifth International Workshop on Security (IWSEC 2010): 251–266.

VII Halunen K (2011) Multicollisions and Graph-based Hash Functions. Proceedings of the Third International Conference on Trusted Systems (INTRUST 2011): 156–167.

Contents

1 Introduction

The need to communicate securely and reliably in a possibly insecure and unreliable environment has existed for as long as information has been perceived to have any significant value. In recent years, the amount of communication has increased tremendously with the advent of the Internet and other electronic communication media. Together with the new, electronic nature of communication it has become evident that many of the traditional means to achieving security and reliability in communications are no longer effective.

There are two branches of mathematics, which try to answer the theoretical and practical problems posed by the new challenges in communication. *Coding theory* seeks answers to the reliability side of the communication, i.e. how can one transmit information across noisy channels efficiently and without errors. *Cryptography* examines the possibilities of securing the communications from eavesdropping and other misuse. This field is quite old with some methods dating back to ancient Greece and Babylon, but nowadays, cryptography has become more and more pervasive in communications systems and it has also branched in new directions such as secure multi-party communication, secret sharing schemes, e-voting and zero-knowledge proofs.

One problem, which has been fairly easy with handwritten correspondence, is the problem of authenticating the source of the message and making sure that the message has not been altered en route to the recipient. With a handwritten letter, the sender can authenticate herself by signing the letter and protect the message from being switched or altered during transmission, by placing the letter in a sealed envelope. Of course, these methods are not perfect, but the problem is more complex as for digital messages. With digital messages, such as email, the message is displayed on the screen and usually any message of valid form will be displayed. Thus, the recipient of an email cannot be sure whether the signer is actually the real sender or if the message has been modified on the way to the recipient's display.

Hash functions are one method for solving the cryptographic problem mentioned above. A hash function is an algorithm that takes as input any message and computes a unique value of prespecified length for that message. This value is known as the *hash value* or "fingerprint" of the said message. With the help of these hash functions, one

can perform message authentication and digital signatures, which are crucial in modern information technology. These functions can also be used in digital timestamping schemes and pseudorandom number generation.

For this thesis, we had two main paths of research. In the first one, we analysed the cryptographic properties of the Very Smooth Hash (VSH) hash function. In the second, we examined the properties of iterated and generalised iterated hash functions against multicollisions. The contributions of this thesis are described in detail below.

1.1 Contributions of this thesis

The main contributions of this thesis are the following:

- A practical study of the preimage resistance of the VSH hash function in a password setting and an improvement of the previous preimage finding method by Saarinen (2006).
- Extending the preimage attacks to discrete logarithm variants of VSH.
- Development of a unified theoretical framework for studying (generalised) iterated hash functions.
- Using the framework to prove new results on the security of iterated hash functions, e.g. showing that arbitrarily large multicollisions can be found faster than with previous methods and for a larger family of iterated hash functions than before.
- Showing that the previous methods can be extended to some graph-based hash functions.

This thesis consists of seven publications.

In Publications I, II and VI, we consider the security of the VSH hash function and its different variants against preimage attacks. In I, we show that the original preimage attack by Saarinen (2006) can be used in a realistic setting with 8-character passwords. In II, we modify the method of Saarinen (2006) in a way that facilitates the reuse of the precomputed tables and thus makes finding multiple preimages easier. In VI, we demonstrate that these preimage finding methods generalise to the discrete logarithm variants of VSH.

In Publications III–V and VII, we demonstrate the results of the research on the multicollisions in generalised iterated hash functions. In III, we give some preliminary results on the Nested Multicollision Attack Schema (NMCAS). In IV, we demonstrate a new method for finding multicollisions in iterated hash functions and show that this is

better than the method by Joux (2004) in some cases. We also describe a general limit for the complexity of finding arbitrarily large multicollisions for iterated hash functions. In V, we detail the NMCAS with exact proofs and evaluate its complexity. Publication VII continues the work on multicollisions in generalised iterated hash functions by generalising the results of III and V to some graph-based hash functions.

In Publications I and II, the study was planned and conducted mainly by the author and the discussion and conclusion were mostly the author's own contribution. The programming and the practical results were done in cooperation with M.Sc. Pauli Rikula, based on some earlier programs written by the author. In Publications III and V, the main contribution was made by Docent Juha Kortelainen and the author was mainly responsible for the background work on iterated hash functions and parts of the discussion, conclusions and interpretation of the theoretical results with some minor contribution on the theoretical findings on multicollisions. In IV, the main result was due to M.Sc. Tuomas Kortelainen with the author of this thesis providing some input on the theoretical limit result and in the background and discussion sections. In Publication VI, the author was responsible for both the theoretical results and for conducting the practical experiments partially based on the work done in Publications I and II. In VII, the results are by the author of this thesis.

2 Applications of hash functions

Before going into the details of modern cryptographic hash function design and the security of hash functions, we give some examples of the applications of hash functions. These examples serve as a particular motivation for this thesis.

Hash functions have become an integral part of many cryptographic protocols and they can also be applied to some non-cryptographic problems. In these cases, some of the security properties required for cryptographic hash functions are not necessary for the hash function. Some of the methods presented below can be considered "folklore" in cryptography and thus no attribution has been made in the examples. More thorough presentation on these topics can be found in the thesis by Van Rompay (2004) and in the Handbook of Applied Cryptography (Menezes *et al.* 1996).

2.1 Data integrity

Hash functions can be applied to check the integrity of data sent over an untrusted channel. If there exists another, trusted channel for sending small messages, the following simple example shows how to use a hash function to check the integrity of the message. Assume that h is a hash function and x a message that Alice wants to send to Bob. Hash functions are usually assumed to be public, so that anyone can easily compute the hash value of any given message. Now, Alice and Bob may use h to ensure the data integrity of x in the following manner:

Alice uses h and computes $h(x)$. Then she sends the message x to Bob over the insecure channel and the value $h(x)$ over the trusted channel. When Bob receives the message x' from the untrusted channel, he computes $h(x')$. If $h(x') = h(x)$, Bob can be assured that the message he received was the one that Alice sent, if the hash function h is cryptographically secure.

Of course, if the untrusted channel is used to send both the message and the hash value, one can always intercept the message sent by Alice, replace it with another and send this message with the corresponding hash value to Bob. Bob would not know that this is not the original message, unless he can somehow communicate with Alice or unless some other, more complicated protocol is used.

In practice, cryptographic hash functions are used for example to check the integrity of program files downloaded from the Internet. In the simplest case, this means that the hosting site provides the hash value of the file (under some publicly known, secure hash function) on their site and when one downloads the file one can check whether the hash values match. This, of course, does not offer much added security if the files and hash values are distributed over the same channel, but is still used as an easy safeguard against rogue code. In order to gain more security, one could check the hash value over some other, trusted channel as in the above example. However, code authenticity is usually verified by code signing in many modern software products. This requires the use of public key cryptography techniques and the distribution of the required public keys to the users of the software.

2.2 Authentication and encryption

By authentication of a message, we mean that the receiver can be sure the message comes from a legitimate sender and that the message itself has not been changed during transmission. It can also happen that even an encrypted message can be changed during transmission (without knowing the key) and that the resulting ciphertext will be decrypted into a plaintext message. Clearly, the above procedure cannot achieve security against these threats if only an untrusted channel is used. With the help of symmetric encryption, this can be achieved.

We assume that E_k is an encryption function that uses the (secret) key k and can be applied to encrypt any given message by the parties that know the secret key. Also it can be reversed to obtain the plaintext, if the key is known. Now, Alice and Bob can proceed as follows: We assume that Alice and Bob have previously agreed on some secret keys k_1 and k_2 that they will use when encrypting and authenticating the message.

Alice takes the message x and encrypts it with E_{k_1} to obtain $C = E_{k_1}(x)$. Then Alice computes a message authentication code (MAC) over C (using k_2) to obtain the MAC value t. Now, Alice sends Bob $C||t$. Bob first verifies the authenticity of the received message and then decrypts, if the check is passed. This way, the authenticity of the message is assured.

There are three different possibilities to combine the encryption and the computation of the MAC. The above method is known as *encrypt-then-MAC*. The two others are *MAC-then-encrypt* and *encrypt-MAC-plaintext*. According to Bellare & Namprempre

(2000), the first is secure in most of the different scenarios that they propose, whereas the other two display weaknesses in many of these scenarios.

Realising a MAC can be done in different ways, but one widely used method is *Hash-based Message Authentication Code, (HMAC)*, where a hash function is used to build the MAC value. This hash function can be any hash function (e.g. MD5, SHA-1). The security of the HMAC is related to that of the underlying hash function and thus designing secure cryptographic hash functions is essential for the security of HMAC message authentication.

2.3 Applications as one-way functions

Hash functions can also be applied as one-way functions, for example when someone wants to commit to a message of certain content and later prove that the message content was the same as in the original commitment. In a simplistic manner, these could be used also to prove some knowledge without disclosing the knowledge itself. For example, Alice might want to convince Bob that she knows the winning numbers of a national lottery in advance.

First, Alice and Bob agree on some coding scheme (e.g. ASCII) to represent the numbers and the ordering of the numbers. Then, Alice generates a random string of appropriate size (say, 256-bits). Now, Alice can compute the hash value of this random string catenated with her intended message and give the resulting value to Bob. After the lottery, Alice makes the random string public and Bob can compute the hash value of the winning numbers and compare that with the one given to him by Alice. If the values agree, Bob can be assured that Alice knew the right values. On the other hand, because of the one-way properties of the hash function, Bob cannot obtain the right numbers from the hash value. The random string is necessary to prevent Bob from making guesses at the numbers before the lottery.

One-way functions are also useful in modern information security contexts. For example, many operating systems store password information in hashed form. This means that the plaintext of the password is not stored in the file, but the hash value of the password instead. When a user enters her password to the system, the system computes the hash value and compares that with the stored value. If the password file gets into the wrong hands, the attacker will not automatically gain the password information, but only the hash values, which should be hard to invert.

Passwords stored in this fashion are susceptible to the so-called *dictionary attack*. This means that the attacker computes the hash values of passwords in some small subset of all the possible passwords. Then, the attacker compares the values in the database with the precomputed values in his dictionary. If a match is found, the attacker learns the password. The success of this attack is based on the observation that when people choose their passwords, they usually try to come up with something that is easily memorised and thus choose passwords from a relatively small subset of all the possible passwords. Thus, a dictionary attack can be very effective even when the one-way function is secure. This attack can be alleviated by adding a *salt* (a short random string) to the beginning of the password before hashing it. This salt can be made public and it makes dictionary attacks much harder for the attacker as the attacker would need to build a different dictionary for all possible salts.

In general, there are many methods for inverting these one-way functions that use the property of *time-memory trade-off* (Hellman 1980). This means that the cost of an attack is divided between the time of computing the hash values and then making look-ups on some precomputed table. In an extreme case, the attacker could build a dictionary for all the possible passwords. Then, the attacker would need only single table look-up to invert the hash value of any given password. Of course, this should be infeasible, when the passwords are chosen from a large alphabet and are long. However, there is a possibility for the attacker to make an optimisation based on his resources available for time and memory. Barkan *et al.* (2006) give a thorough analysis on the bounds of time-memory trade-off attacks.

Also pseudorandom sequences can be generated with one-way functions. In this type of scenario, one is given a seed number s and a one-way function f. It is then possible to construct a sequence $f(s), f(s+1), f(s+2), \ldots$. The seed s should be a secret and the function f publicly known. If the one-way function f could be inverted, then, the sequence could be predicted and the randomness of the sequence would be lost. Hash functions can be used in this pseudorandom sequence generation when they meet the one-wayness requirement.

2.4 Digital signatures

In modern commerce, electronic transactions have become more and more commonplace. Thus, there is a need to have the digital equivalents of signatures. The advent of public key cryptography has enabled digital signatures, but the signing of a large file or

database is computationally very expensive. There should also be a safeguard against the repudiation of the signed document. Thus, cryptographic hash functions are used and the hash values are signed instead of the possibly very large files or messages. This way, better performance is achieved, as well as security against possible attacks such as forgery and repudiation.

The basic strategy for digital signatures is the following: First, Alice generates a key pair (S_A, P_A), where S_A is Alice's secret key and P_A her public key. When Alice wants to sign a message x, she computes the hash value $h(x)$ and then inputs $h(x)$ into the signing algorithm, which depends on the secret key S_A. The result $s(x) = \text{sign}(S_A, x)$ is then sent to Bob, who can verify the signature by using a verification algorithm v. This algorithm uses the public key P_A, the hash value of the signed message and the received signature (and possibly some other data) and computes $v(s(x)) = \text{ver}(P_A, s(x))$.

The basic principles of this type of use for hash functions have been laid by Diffie & Hellman (1976), Rivest *et al.* (1978), Lamport (1979) and Merkle (1989a). A more recent study on digital signatures has been made by Lysyanskaya (2002). Nowadays, digital signatures are considered as reliable and binding as pen and paper signatures in many countries. There are also several standards for digital signatures, e.g. FIPS 186-3 (2009), ISO/IEC 9796-2 (2010).

2.5 Digital timestamping

Sometimes, it is necessary to establish the timeliness of communications, documents, transcripts or other records. For this, one needs digital timestamps. These can be used to establish either the relative timeline for a sequence of messages, transactions or documents or an absolute timeline. A relative timeline means that the order of the messages can be verified, but the exact times of the messages cannot be recovered. An absolute timeline can be used to ascertain the exact times of the messages. In public key cryptosystems, timestamps are important because they can be used to determine whether a message was signed before a key had been revoked.

Hash functions can be used in both of these schemes. First, we assume that there is a Time Stamping Authority (TSA), which is a trusted party, who issues the timestamps for the messages. Now, Alice can have message x timestamped by computing $h(x)$ and sending her identity information (ID_A) to the TSA, together with a serial number n for the message and the current time t_n. The TSA then forms the timestamp by signing these values. Thus, timestamp $= \text{sign}(ID_A, h(x), n, t_n)$ and this value is returned to Alice

as the timestamp. To verify the timeliness of the message x, one can use the public key of the TSA to verify the signature and then compare the hash value in the timestamp with the hash value of the message the timestamp is attached to.

In the above scheme, the timestamp holds the absolute time t_n and even the TSA does not learn the message x (only its hash value is sent to the TSA). However, a malicious TSA could issue timestamps with false (earlier) time than in reality. To overcome this obstacle, we may use the following scheme: Suppose that Alice wants to timestamp the n^{th} message x_n in some communication. Alice computes $h(x_n)$ and sends this to the TSA. Now, the TSA takes a previous linking value l_{n-1} (for the first message this can be some random string) and computes the value $l_n = h'(h(x_n)\,||\,l_{n-1})$, which becomes the next linking value. Here, h' is a hash function, which may be different from h. The linking values are stored in a database by the TSA and the relative timeline for the timestamped messages can be verified.

The above methods require the hash functions to be collision resistant in order to generate unforgeable timestamps. In the latter method, the TSA cannot forge timestamps as in the first method. On the other hand, the second method only generates relative timestamps. To gain the best of both worlds, one could use both methods. Then, one would have unforgeable, absolute timestamps.

A survey on timestamping methods has been done by Une (2001). The basic ideas behind timestamping schemes have been introduced by Haber & Stornetta (1990) and Bayer *et al.* (1993). The ISO/IEC 18014 (2009) defines a standard for digital timestamping.

3 Basic concepts

In this chapter, we present the basic definitions used throughout this thesis. We also briefly describe some fundamental results concerning iterated hash functions. Furthermore, we present some of the traditional methods for constructing hash functions and give the result of Damgård (1989) concerning the collision resistance of the iterated construction. We also briefly describe the security notions for (iterated) hash functions and security models at the level that is necessary for this thesis.

3.1 Words, relations and basic algebra

In order to define hash functions, we need some basic concepts. Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of *natural numbers* and $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$ the set of all positive natural numbers. For each finite set S, let $|S|$ be the cardinality of S, i.e. the number of elements in S.

Now, we say that *an alphabet* is any finite, nonempty set of symbols, which are called *letters*. If A is an alphabet, we say that any finite sequence of letters from A forms *a word* (over A). So, if $w = a_1 a_2 \ldots a_n, n \in \mathbb{N}$ is a word over A, then $a_i \in A$ for all $i = 1, 2, \ldots, n$. The number n is *the length* of the word w and it is denoted by $|w|$. If the length of a word is 0, we say that this word is *the empty word*, which contains no letters. The empty word is unique and it is denoted by ε. By $|w|_a$, we denote the number of occurrences of the letter $a \in A$ in the word w.

Let $n \in \mathbb{N}$ and A be an alphabet. We define the set A^n as the set of all words over A with the length n. We also define $A^* = \bigcup_{i=0}^{\infty} A^i$ and $A^+ = \bigcup_{i=1}^{\infty} A^i$. Notice that $A^+ = A^* \setminus \{\varepsilon\}$. Let $w, u \in A^*$. We define the *catenation* $(\cdot)$ of w and u as $w \cdot u = wu$, i.e. the words w and u written one after the other. Clearly, catenation is a binary operation over A^* (and A^+, if $w, u \in A^+$) and we omit the $\cdot$ when applying catenation on words, e.g. wu means the catenation of w and u. As algebraic structures, $(A^*, \cdot)$ is a *free monoid* and $(A^+, \cdot)$ is a *free semigroup*.

Let $m \in \mathbb{N}$. The word $u = u_1 u_2 \cdots u_m \in A^*$ is *a subword* of the word $w \in A^*$ if there exist $x_0, x_1, \ldots, x_m \in A^*$ such that $w = x_0 u_1 x_1 \ldots u_m x_m$. A subword u of w is a *factor* of w if $w = x_0 u x_1$ for some $x_0, x_1 \in A^*$. We define $\mathrm{alph}(w) = \{a \in A \,|\, |w|_a > 0\}$ as the *alphabet of* w. A *permutation* of an alphabet A is any word $w \in A^+$ such that $|w|_a = 1$ for each $a \in A$. Furthermore, we say that any subset of A^* is *a language* (over A).

Let A and B be alphabets. A mapping $\tau : A^* \to B^*$ is a (*monoid*) *morphism* if $\tau(uv) = \tau(u)\tau(v)$ for each $u, v \in A^*$ and $\tau(\varepsilon) = \varepsilon$. If $B \subseteq A$, then the *projection morphism* from A^* into B^*, denoted by π_B^A (or π_B, when A is clear from the context), is defined by $\pi_B^A(b) = b$ for each $b \in B$ and $\pi_B^A(a) = \varepsilon$ for each $a \in A \setminus B$.

Let L and L' be languages. The *catenation* of L and L' is the language $LL' = \{uv \mid u \in L, v \in L'\}$, which is a logical extension from the catenation of words. We define the *powers* of L recursively as follows: Let $L^1 = L$, and $L^{i+1} = L^i L$ for $i \in \mathbb{N}_+$. The *Kleene*$^+$ of L is the language $L^+ = \cup_{i=1}^{\infty} L^i$. The *catenation closure* of L is the set $L^* = L^+ \cup \{\varepsilon\}$. For the sake of simplicity, we write w^+ instead of $\{w\}^+$, for any word w.

A binary relation R of set X is called *irreflexive*, if for all $x \in X, (x,x) \notin R$. R is called *antisymmetric*, if for all $x, y \in X$, $(x,y) \in R \Rightarrow (y,x) \notin R$ holds. Furthermore, if for all $x, y, z \in X$, $(x,y) \in R, (y,z) \in R \Rightarrow (x,z) \in R$ holds, then R is *transitive*. A binary relation R of the set X is a *partial order* (*in* X) if it is irreflexive, antisymmetric and transitive.

Let $\prec$ be a partial order in X. We say that $(X, \prec)$ is a *partially ordered set* or poset for short. The elements $x, y \in X$ are *incomparable* (in $(X, \prec)$) if neither $x \prec y$ nor $y \prec x$ holds. The nonempty finite sequence $x_1, x_2, \ldots, x_n$ of elements of X is a *chain* of $(X, \prec)$ if $x_i \prec x_{i+1}$ for $i \in \{1, 2, \ldots, n-1\}$. Here, $n \in \mathbb{N}_+$ is the *length* of the chain $x_1 \prec x_2 \cdots \prec x_n$. For each chain c of $(X, \prec)$, let $|c|$ be the length of c. An (indexed) set of chains $\{c_i\}_{i \in I}$ is a *chain decomposition* of $(X, \prec)$, if $\{C_i\}_{i \in I}$ is a partition of X, where $C_i = \{x \in X \mid x \text{ occurs in the chain } c_i\}$. Obviously, a chain decomposition exists for all posets.

Let X be a finite poset, i.e. X is a finite set with some partial order. The *maximum number of incomparable elements* of $(X, \prec)$ is the cardinality of the largest set $Y \subseteq X$ such that the elements of Y are pairwise incomparable. The *minimum chain decomposition size* of $(X, \prec)$ is the smallest number $m \in \mathbb{N}_+$ such that there exist chains $c_1, c_2, \ldots, c_m$ of $(X, \prec)$ for which $\{c_i\}_{i=1}^m$ is a chain decomposition of $(X, \prec)$. Finally, let *maximum chain length* of $(X, \prec)$ be the greatest number $m \in \mathbb{N}_+$ such that there exists a chain of length m in $(X, \prec)$.

An important connection between two of the concepts defined above is stated in a famous theorem of Dilworth (1950). This result will be useful in the study of multicollisions on generalised iterated hash functions (see Chapter 5 of this thesis).

Theorem 1 (**Dilworth's Theorem**). *Let $(X, \prec)$ be a finite, partially ordered set. Then, the maximum number of incomparable elements of $(X, \prec)$ is equal to the minimum chain decomposition size of $(X, \prec)$.*

Let A be an alphabet and $\alpha \in A^*$. We define a partial order $\prec_\alpha$ induced by α in $\text{alph}(\alpha)$ by setting $a \prec_\alpha b$ if and only if all occurrences of a are before the first occurrence of b in the word α, for all $a, b \in \text{alph}(\alpha)$. We call the elements $a_1, a_2, \ldots, a_l \in \text{alph}(\alpha)$ *independent* (with respect to $\prec_\alpha$) if these elements form a chain in the poset $(\text{alph}(\alpha), \prec_\alpha)$.

3.2 Finite fields and groups

In order to understand the notion of the discrete logarithm problem (DLP) and its variations and generalisations, we need to define some basic concepts of group theory. First, assume that G is a finite, cyclic group, with a generator g. We denote this by $G = \langle g \rangle$.

Definition 1. Let $G = \langle g \rangle$, $|G| = n$ and $z \in G$, drawn uniformly and at random from G. Now, finding the unique value $t \in \{0, 1, \ldots, n-1\}$ for which $z = g^t$, is the discrete logarithm problem (DLP) in G.

The value t in Definition 1 is called *the discrete logarithm* of z in the base g. The DLP is a basis for many modern cryptosystems, especially in public key cryptography. For example, the Diffie-Hellman key exchange protocol is based on the difficulty of solving the computational DLP (Diffie & Hellman 1976). The computational DLP states that it is computationally intractable to compute g^{ab}, when only g, g^a and g^b are given. There are also several other variations of the DLP, which have been used in cryptography. Lenstra *et al.* (2006) give a definition for the k-modified DLP, which is used in one of their variants for the Very Smooth Hash (VSH). This is needed in Chapter 4, where we show our results on the cryptanalysis of VSH.

Definition 2. Let G be a finite cyclic group with generator g and $k \in \mathbb{N}_+$. Let C be a subgroup of G with a generator $c = g^{\frac{|G|}{|C|}}$. Furthermore, let $f : G \to G$ be a mapping such that $f(g_i)^{\frac{|G|}{|C|}} = g_i^{\frac{|G|}{|C|}}$ for all $g_i \in G$. The k-modified discrete logarithm problem for G, C and f, is to find a nonzero solution $(d_1, d_2, \ldots, d_k)$ for

$$\Big(\prod_{i=1}^{k} f(g_i)^{d_i}\Big)^{\frac{|G|}{|C|}} = 1 \tag{1}$$

when the g_i are drawn uniformly and at random from G for $i = 1, 2, \ldots, k$.

Because the DLP and many of its variants require the underlying group to be quite large, finding small groups where the DLP is hard has been an active topic in cryptography. Two very widely used families of groups are the multiplicative groups of finite fields and groups formed by elliptic curves. We denote by $\mathbb{F}_q$ the finite field of order q. The DLP in the multiplicative group $\mathbb{F}_q^*$ has been widely used in cryptography and there are some very efficient methods of computing in and/or compressing the results into some subgroups of these groups, e.g. XTR (Lenstra & Verheul 2000) and CEILIDH (Rubin & Silverberg 2003).

To make the resulting values even shorter, elliptic curves can be utilised. Elliptic curves over prime fields $\mathbb{F}_p, p > 3$ can be represented in the short Weierstrass from as

$$y^2 = x^3 + a_4 x + a_6, \tag{2}$$

where one usually chooses $a_4 = -3$ to extract some performance benefits in the arithmetic (Lenstra *et al.* 2006). The set of points (x, y) that satisfy (2) and a point at infinity form an Abelian group when the addition is defined appropriately. The point at infinity is the identity element and the inverse of a point (x, y) is $(x, -y)$. Elliptic curves require usually less bits for the same level of security than finite fields or other groups where the DLP is hard.

3.3 Graphs

In Chapter 5, we also study graph-based hash functions, which are a generalisation of iterated hash functions. Here, we give some basic definitions of graph theory that are used in the study of graph-based hash functions.

A *graph* $G = (V, E)$ consists of a finite set of *vertices* V and the set $E \subseteq V \times V$ of *edges* between vertices. A *path* between two vertices u and v is a sequence of vertices $(u, w_1, w_2, \ldots, w_t, v)$ with an edge from each of the vertices to the next vertex in the sequence. A *cycle* is a path from v to v. A graph is *connected* if for each pair of vertices u, v there exists a path between them. A graph G is a *tree* if it is connected and has no cycles.

In the following, we consider only directed graphs (digraphs for short). In a digraph, the *indegree* of a vertex v is the number of edges that have v as an endpoint and the *outdegree* of v is the number of edges that have v as a starting point. A *source* of a

digraph is any vertex that has an indegree equal to zero and a *sink* is a vertex with an outdegree equal to zero. In a tree, sources are called *leaves* and sinks are called *roots* of the tree.

A t-ary tree G is a tree for which all vertices have an indegree t or 0, an outdegree equal to one and there is a unique sink r called the root of G. All non-leaf, non-root vertices are known as intermediate vertices. A 2-ary tree is also called a *binary* tree. For a more thorough presentation on graphs and their properties, see the book by Diestel (2006).

Let $G = (V,E)$ be a digraph. For any vertex v in G, we define $u \to v$ iff $(u,v) \in E$. Furthermore, we denote by $u \Rightarrow v$ if there exists a directed path from u to v or if $u = v$. If $u \Rightarrow v$, we say that v is *reachable* from u. Let $v \in V$. We denote by $G[v] = (V[v], E[v])$ the subgraph of G such that $V[v] = \{u \in V : u \Rightarrow v\}$ and $E[v] = \{(u,v) \in E : u,v \in V[v]\}$. Finally, we denote by $L(G)$ the set of sources of G and $L(v) := L(G[v])$ for any $v \in V$.

3.4 Hash functions

Now, we can define hash functions as mathematical entities. It is well known that any message can be presented as a word over the binary alphabet $\{0,1\}$, so we may assume that all the messages are from $\{0,1\}^*$.

Definition 3. A *hash function* is any function $f : \{0,1\}^* \to \{0,1\}^n$, where $n \in \mathbb{N}_+$. The number n is the length of f.

As can be seen from the above definition, there are many functions, which qualify as hash functions. For example, for all $m \in \mathbb{N}_+$, the function $f(x) = x \mod m$, where $x \in \{0,1\}^*$ is a natural number understood to be in the binary form, is a hash function of length $\lceil \log_2 m \rceil$ when we interpret the residues modulo m as binary numbers. Thus, we can easily devise hash functions satisfying the very general definition above.

However, there are some properties that a hash function must possess in order to be considered as a cryptographically useful hash function. These properties have been studied quite extensively and we present only the most commonly used definitions for these properties. There are some other uses for hash functions, for which there is no need for a hash function to possess these cryptographic qualities, e.g. using a hash function to generate a hash table for fast table look-ups.

In general, we use asymptotic notation in the evaluation of the security of hash functions. Let f and g be functions from $\mathbb{N} \to \mathbb{N}$. We say that a function f is

asymptotically smaller than g (denoted by $f \in O(g)$) if there exists $n_0 \in \mathbb{N}_+$ and a constant $c > 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. This means that for large values of x, $f(x)$ is bounded from above by $g(x)$. Furthermore, we say that function f is asymptotically greater than g (denoted by $f \in \Omega(g)$) if there exists $n_0 \in \mathbb{N}_+$ and a constant $c > 0$ such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$. If $f \in O(g)$ and $f \in \Omega(g)$, we say that f is asymptotically equal to g and denote this by $f \in \Theta(g)$.

The following are some informal criteria for cryptographic hash functions:

- The hash function must accept messages of any length as its input.
- The value of the hash function has to be easily computable.
- The hash function should be one-way, i.e. given $y = f(x)$ and f, it should be hard to obtain x.

Any function that has the above three properties is called *a one-way hash function*. If, in addition to the above, it is hard to find any two distinct messages x and y satisfying $f(x) = f(y)$, the hash function is called *collision resistant*. These properties have already been proposed in the thesis by Merkle (Merkle 1979, pp. 12-13) and formalised in several publications afterwards (e.g. Rogaway & Shrimpton (2004), Andreeva & Stam (2011)).

In the Handbook of Applied Cryptography (Menezes *et al.* 1996, p. 323), there are definitions for three properties of hash functions. These properties are quite similar to those presented above.

- *Preimage resistance* means that for essentially all prespecified outputs it is computationally hard to find any input, which hashes to that output.
- 2^{nd} *preimage resistance* means that for any given input it is computationally hard to find a second input with the same output.
- *Collision resistance* means that it is computationally hard to find any two distinct inputs with the same output.

These properties are usually required for hash functions to be considered applicable in a cryptographic context.

Of course, defining the word “hard” in this context is the key to hash function security. For example Menezes *et al.* (1996), do not give any strict definition of “hard”, but state instead that it is very dependent on the context in which the hash functions are

used. However, there are some measures of this "hardness" that are widely accepted to be the benchmark for hash functions. These are discussed later.

More refined analysis of the security properties of hash functions has been done by Rogaway & Shrimpton (2004), who take a more granular look at preimage and collision searches. They show that between the seven security properties that they define there are implications and separations. Thus, some of these properties are more desirable as they imply others for the hash function. There are also security properties, which are quite independent of the others. Especially, the notion of *keying* hash functions is very useful as some security results require that the hash function is chosen from a family of hash functions. Keying means that the hash function takes the key as an extra input and this key is used to differentiate the hash functions of the same family from each other.

Some additional results regarding these more refined security notions have been presented by Andreeva & Stam (2011). In this thesis, we concentrate on the more general aspects of preimage and collision resistance and do not take full advantage of these results.

The formal model of an *ideal hash function* (of length n) is the *random oracle* (of length n), which is a function, that takes as input any message and chooses the hash value uniformly at random from $\{0,1\}^n$. This is also called a variable input length random oracle (VIL-RO). However, this type of ideal random oracle cannot be efficiently realised and thus these cannot be utilised in constructing hash functions in the real world. However, these ideals have been used to prove the security of cryptographic protocols and real hash functions, such as iterated hash functions, are used instead of the ideal for practical purposes.

For an ideal hash function h of length n, we expect the following security properties. Finding a preimage x of any $y = h(x)$ should require the computation of approximately 2^n messages, i.e. the function describing the complexity of finding the preimage should be in $\Theta(2^n)$. The same holds for second preimages, meaning that given x and $h(x)$ it should require $\Theta(2^n)$ computations to find $y \neq x$ for which $h(x) = h(y)$. The so-called *birthday paradox* states that it is likely that in a small group of some tens of people there are two with the same birthday, even though there are 366 possible birthdays. From this observation, it can be shown that finding two distinct messages x and y with $h(x) = h(y)$ should have complexity in $\Theta(2^{\frac{n}{2}})$. This can be done simply by computing and listing hash values until a collision is found. This is known as the *birthday attack* against hash functions. The above are also the limits of brute force attacks against particular hash functions.

An attack against a hash function is a (usually probabilistic) method for reducing the complexity of finding a preimage, a second preimage or a collision. This means that one demonstrates that the complexity of the particular attack is bounded from above by some smaller function than in the case of an ideal hash function. Sometimes, this reduction is merely theoretical in the sense that even with the attack method at hand, it is still infeasible to produce a (second) preimage or a collision, even if the complexity is lower than for an ideal hash function. *A practical attack* is an attack by which one actually provides the messages that are (second) preimages or collisions for the given hash function. Nowadays, many collision attacks against particular hash functions are practical, e.g. MD4 and MD5, whereas most preimage finding attacks against hash functions are mostly theoretical.

Attacks against hash functions can also be roughly divided into two other categories besides theoretical and practical, namely *general attacks* and *special attacks*. An attack in the latter category concerns only a single hash function, such as MD5 or VSH. In this type of attack, usually some weakness of the specific underlying function is used to gain advantage over the brute force attacks. A general attack applies to a larger set of hash functions, such as all iterated hash functions, and is thus applicable to a wide range of hash functions. These attacks usually assume that the underlying primitives are ideal and show weaknesses in the actual method of compression instead of weaknesses in some part of a specific function.

3.5 Constructions

As mentioned in the previous section, there is no known method for efficiently constructing an ideal hash function. However, there are methods for constructing hash functions with at least some of the desired properties. The most popular method has been the idea of *iterated hash functions* devised by Rabin (1978). In order to describe the method, we need one further definition.

Definition 4. Let $n, m \in \mathbb{N}_+$. A *compression function (of length n and block length m)* is any function $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$.

An ideal compression function can be described as a fixed input length random oracle (FIL-RO) (of length n). A FIL-RO is a function that accepts two inputs: one is any message of length m and the other is any message of length n. The output is chosen from $\{0,1\}^n$ uniformly at random. Again, it is not possible to efficiently implement a

FIL-RO. For a FIL-RO, the same limits to finding preimages, second preimages and collisions apply as for a VIL-RO.

Let $m,n \in \mathbb{N}_+$ and $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ be a given compression function. We define the function $f^* : \{0,1\}^n \times (\{0,1\}^m)^* \to \{0,1\}^n$ inductively as follows. Let $y_0 \in \{0,1\}^n$, $y_1 \in (\{0,1\}^m)^*$, and $y_2 \in \{0,1\}^m$. Then, $f^*(y_0,\varepsilon) = y_0$ and $f^*(y_0,y_1y_2) = f(f^*(y_0,y_1),y_2)$.

The idea of an iterated hash function is quite simple. Let f be as in Definition 4 and $y_0 \in \{0,1\}^n$. Now, we may take any message x from $\{0,1\}^*$, divide x into d blocks of length m ($x = x_1x_2\cdots x_d$), add zeroes to the end of the message if the length of the message is not divisible by m, add also a block denoting the length of the message as the final block x_{d+1} and compute in the following iterative manner

$$y_i = f(y_{i-1},x_i), \text{ for } i = 1,2,\ldots,d+1.$$

Now, the final value of the iteration y_{d+1} is then defined as the hash value of the message x. If we define $x' = x_1x_2\cdots x_{d+1}$, then the hash value is $h(x) = f^*(y_0,x')$.

Now, this construction does not automatically guarantee any of the necessary security properties for hash functions. The results of Merkle (1989b) and Damgård (1989) state the following:

Theorem 2. *Let f be a compression function of length n. If f is collision resistant, then the iterated hash function h constructed from f is also collision resistant.*

Theorem 2 holds for any suffix-free set of messages. The proof by Damgård (1989) is based on the idea that any collision on h can be reduced to either a collision on f or to imply that the two colliding messages are the same. Both of these are, of course, contradictory to the assumptions of the theorem. Merkle (1989b) shows essentially the same result. There is one drawback in the original method presented by Damgård, which requires one to sacrifice one bit for each message block in order to get a suffix-free encoding of the messages. There is a workaround for this, where one just adds the length of the message at the end of the message as a part of the final block as we did in our construction above. This ensures the suffix-freeness of the messages and thus allows the above result to be applied. This limits the length of the message, but this limit is usually large enough to accommodate all practical messages (2^{128} bits or more). Figure 1 presents the basic MD-construction for iterated hash functions.

Damgård (1989) also assumes that the compression function is drawn from a family of possible compression functions. In real constructions, the compression function is

usually fixed. This issue is tackled in the research of Rogaway & Shrimpton (2004) and Andreeva & Stam (2011) by considering the properties against keyed and unkeyed hash functions separately. This yields a more refined analysis of the different security properties.

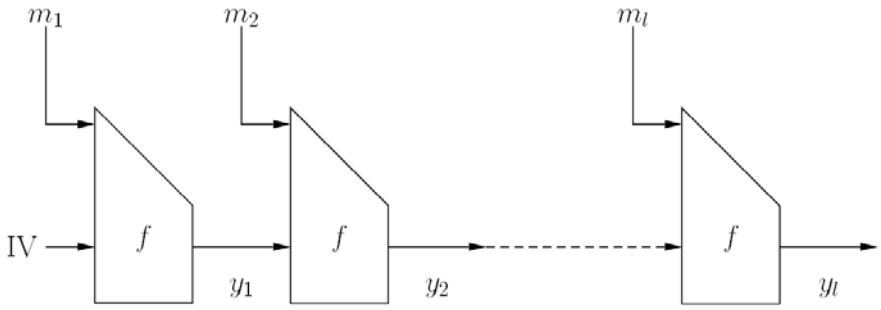

Fig 1. Basic MD-type iteration.

Due to Theorem 2, collision resistant hash functions could be designed, if collision resistant compression functions can be devised. However, proving the compression function collision resistant with mathematical certainty has been a very difficult task and most of the modern hash functions have quite varied arguments on the security of the compression function or the hash function as a whole. The security proofs are usually quite involved and hardly ever can one directly prove that the compression function is actually collision resistant.

One should also note that even though the collision resistance of the underlying hash function is sufficient for the iterated hash function to be collision resistant, it is not a necessary condition. One example of this is Very Smooth Hash (VSH), which has a very rigorous proof of its collision resistance, even though the underlying compression function is not collision resistant (Contini *et al.* 2006). VSH is discussed in more detail in Chapter 4 of this thesis.

The results of Preneel (1993) show how the result of Damgård (1989) can be applied to constructions using block ciphers. The results of Black *et al.* (2002) show also the limitations of Theorem 2 as they are able to prove that some of the iterated hash functions with weak (against collisions) compression functions considered already by Preneel (1993) are secure against collisions.

Usually, the complexity of attacks against iterated hash functions is not measured as the number of messages needed to hash in order to gain the desired result. For

the iterated hash functions, the measure of complexity is the number of compression function calls needed for the attack to work. It is easy to see that also with this notion of complexity it takes $\Theta(2^n)$ compression function calls to find a (second) preimage and $\Theta(2^{\frac{n}{2}})$ compression function calls to find a collision for an iterated hash function with a FIL-RO as a compression function with a brute force search.

3.6 Practical implementations and results

The iterated structure described in the previous section prompted the design of many hash functions that worked on that principle. One of the most widely adopted hash functions was MD5 (Rivest 1992), the successor of MD4 (Rivest 1991).

MD5 is still widely used even though there are several flaws found in the design and even very practical attacks have been published (Stevens 2006, Klima 2005, Wang & Yu 2005, Stevens *et al.* 2007). These attacks demonstrate collisions for the hash function and some applications of these collisions in communication protocols. For MD4, collisions were first demonstrated by Dobbertin (1998) and nowadays there are very effective methods for finding collisions for MD4 and MD5. For MD4, there is also a method for finding preimages of the hash values (Leurent 2008), which is of course much harder than finding collisions. Also multicollisions against the compression function of MD4 can be found (Yu & Wang 2007).

The SHA-family of hash functions has been proposed as a more secure alternative to the MD-family of hash functions, but even SHA-1 has been found quite weak against collisions (Biham *et al.* 2005, Wang *et al.* 2005). Thus, the National Institute for Standards in Technology (NIST) has called for a new secure hash function standard for SHA-3 (Kayser 2007). An excellent survey of the multitude of protocols, attacks and the motivation for the competition has been made by Preneel (2008).

The above constructions of MD and SHA hash function families are so called *dedicated hash functions*. In dedicated hash functions, the compression function is designed solely to be used in iterated hash functions. However, these were not the first types of iterated hash functions to emerge. The first iterated hash functions operated on the idea that already known *block ciphers* (such as DES or AES) could be used as compression functions. The distinction between the two ways of constructing hash functions is not very strict as the compression functions of many dedicated hash functions include a part that acts as a block cipher.

There are many ways to construct iterated hash functions from block ciphers. As pointed out by Preneel *et al.* (1993), some of these are secure against collision and/or preimage attacks and some constructions are not. One of the first ideas was the Davies-Mayer -construction, which is the following: Let E_k be a block cipher, i.e, $E_k(x) = y$, where x is a block of plaintext, y is then a block of ciphertext and k a key from some keyspace. Now, Davies-Meyer -method for constructing an iterated hash function is

$$y_{i+1} = E_{x_i}(y_i) \oplus y_i,$$

where x_i is a message block of the same length as the key, y_0 is some given initial value and $\oplus$ denotes the XOR operation. A variation of this is the Matyas-Meyer-Oseas -method, which defines the iteration as

$$y_{i+1} = E_{z(y_i)}(x_i) \oplus x_i,$$

where x_i is as above and z is a function which transforms ciphertext blocks into valid keys for the cipher (if necessary).

The results of Preneel *et al.* (1993) describe many other possible variations of these primitives, and as mentioned above, describe their security in a detailed fashion. However, in this thesis, we will consider a black box approach to the underlying primitive and assume that the compression function f used in the iterative structure is a FIL-RO.

3.7 Security models for hash functions

The security models in which hash function constructions from some underlying primitive (usually a compression function) are studied are basically of two different types. First, one may look at *property preservation* of the type of construction. The idea in this type of reasoning is very simple. If the underlying primitive has some desirable property, then under some assumptions the iterated construction based on this primitive should have the same property. For example, the results of Merkle (1989b) and Damgård (1989) show, that MD-type of iteration preserves collision resistance of the underlying compression function. There are results concerning MD-type of iteration and property preservation, which show that the MD-type of iteration is far from perfect and in fact does not preserve some of the desirable properties for an ideal hash function (see for example the papers by Bellare & Rogaway (1993) and Coron *et al.* (2005)).

Furthermore, Andreeva *et al.* (2007) have studied many iteration methods used in modern hash function designs, including the MD-type iteration. The results show that from the seven properties proposed by Rogaway & Shrimpton (2004) many designs preserve only one or two properties. Especially, the MD-type iteration does not preserve the basic notions of preimage and second preimage resistance. Andreeva *et al.* (2007) propose a new design called ROX and prove that it preserves all the seven security notions of Rogaway & Shrimpton (2004).

Another security model under which iterated constructions can be studied is *indifferentiability (from a random oracle)* introduced by Maurer *et al.* (2004) and Coron *et al.* (2005). In this methodology, one tries to show, that the construction is indifferentiable from a random oracle (or some other idealised function). This way, one can be more convinced of the usability and security of the hash function, because many security proofs for the protocols rely on the assumption that the hash function is a random oracle. Thus, indifferentiability from a random oracle is a very strong security notion. However, Coron *et al.* (2005) show that ordinary MD-type iteration does not meet the requirements of indifferentiability. They present three different ways to overcome this and prove that these three modified methods are in fact indifferentiable from a random oracle. Although indifferentiability from a random oracle is a very desirable property for a hash function, Ristenpart *et al.* (2011) show that indifferentiability is not enough to guarantee the security of schemes, where there are multiple disjoint adversarial stages.

Furthermore, Canetti *et al.* (1998) show that there are some cryptographic protocols, which are proven secure in the random oracle model, but are completely insecure, when the random oracle is replaced by some other, weaker construction. The results of Maurer *et al.* (2004) also lead to this conclusion. Therefore, there are also proofs of security in *the standard model*. In the standard model, the security proofs state that an adversary gains only an insignificant amount of advantage (usually given as a probability for breaking the scheme) when applying a certain amount of queries.

There is a vast amount of research carried out in the formalisation of different security notions and different types of attacks against hash functions. An interested reader is referred to the Ph.D. thesis of Andreeva (2010), where many of these aspects are presented in more detail and the properties of hash functions are extensively studied.

4 Cryptanalysis of the Very Smooth Hash

In this chapter, we describe and analyse the Very Smooth Hash (VSH) hash function, presented by Contini *et al.* (2006). This proposal has very nice mathematical properties, which also enable a proof for its collision resistance. Unfortunately, the preimage resistance of this hash function suffers from these mathematical properties as demonstrated first by Saarinen (2006). We present our improvements to this method and generalise it to some variants of VSH. We also give some results of our experiments from Publications I, II and VI.

4.1 The Very Smooth Hash algorithm

In order to understand the VSH algorithm and prove its collision resistance, we need to define some mathematical concepts related to number theory. First of all, we enumerate the prime numbers and denote by p_i the i^{th} prime number. Thus, $p_1 = 2, p_2 = 3, p_3 = 5 \ldots$ and we define $p_0 = -1$.

Let a and x be integers, $c > 0$ a constant and N a hard-to-factor integer. If a has all its prime factors less or equal to p_k, for some $k \in \mathbb{N}_+$, we say that a is p_k-*smooth*. If $a \equiv x^2 \mod N$ and the largest prime factor of a is at most $(\log N)^c$, we say that a is a *very smooth quadratic residue modulo N*.

It is a well-known fact that all positive integers can be uniquely represented as products of prime numbers. Finding this prime factor representation for a given number is considered a very hard computational task when the number is sufficiently large and this problem is the foundation of many cryptographic techniques, e.g. RSA (Rivest *et al.* 1978). Many algorithms have been developed to achieve fast factorisation of large numbers although a polynomial time algorithm for factorisation has not been devised yet (see for example the book by Crandall & Pomerance (2001)).

The factorisation methods, which are most related to VSH, are the quadratic sieve (QS) (Pomerance 1985) and the number field sieve (NFS) (Lenstra *et al.* 1990). NFS is acknowledged as the fastest method for factoring large integers and the theory behind the algorithm is quite involved. NFS uses many of the same ideas as QS and these ideas are also behind the security of VSH against collisions.

The main connecting idea between QS, NFS and VSH is the use of *relations* of very smooth numbers to factor large integers. A relation in this sense is a congruence of the form

$$v^2 \equiv \prod_{0 \leq i \leq u} p_i^{e_i(v)} \mod N,$$

where $u > 0$ and v are integers and $e_i(v)$ is a $(u+1)$ -dimensional integer vector. If there are more than $u+1$ relations, one can combine the information from these relations with the help of some linear algebra. Specifically, the information can be used to form a relation with all even exponents. Thus, one gains a congruence of the form

$$x^2 \equiv y^2 \mod N.$$

After this, one may find nontrivial factors of N from $gcd(x \pm y, N)$, provided that $x \not\equiv \pm y$ (Contini *et al.* 2006).

Relations can be easily constructed when the factorisation of N is known. One merely computes square roots modulo each of the prime factors and combines these with the help of the Chinese Remainder Theorem (Crandall & Pomerance 2001, p. 81). For a more thorough presentation on prime numbers and factoring algorithms, the reader is instructed to consult the book by Crandall & Pomerance (2001).

For evaluating the security of the VSH algorithm, we need the function, which describes the running time of the NFS. As in the paper by Contini *et al.* (2006), we have

$$L[N,t] = e^{(t+o(1))(\ln N)^{1/3}(\ln \ln N)^{2/3}},$$

for constant $t > 0$ and asymptotically when $N \to \infty$. The NFS expected runtime behaves as $O(L[N, 1.923\ldots])$ (Contini *et al.* 2006).

Furthermore, Contini *et al.* (2006) define a problem under which the security of VSH can be evaluated. This problem is related to finding non-trivial modular square roots modulo a very smooth number.

Definition 5. (Nontrivial Modular Square Root of Very Smooth numbers) Let N be the product of two unknown primes of approximately the same size and let $k \leq (\log n)^c$. The NMSRVS problem is the following: Given N, find $a \in \mathbb{Z}_N^*$ s.t. $a^2 \equiv \prod_{i=0}^{k} p_i^{e_i} \mod N$ and at least one of $e_0, e_1, \ldots, e_k$ is odd.

There are two assumptions on the difficulty of NMSRVS. The NMSRVS assumption states that there is no probabilistic, polynomial time algorithm, which solves the NMSRVS problem with a significant probability. This assumption does not give

guidelines on how to select a suitable modulus for VSH. Thus, there is a need for a computational approach to the problem.

For the other assumption, we need to find the least positive integer S' solving the inequality

$$L[2^{S'}, 1.923\ldots] \geq \frac{L[N, 1.923\ldots]}{u}$$

and assume that finding a relation for this N and u is as hard as factoring an S'-bit integer with NFS. Then, the computational NMSRVS assumption is that solving the NMSRVS problem is as hard as factoring an S'-bit integer. (Contini *et al.* 2006)

The VSH hash function calculates the hash value of a message in the following way (Contini *et al.* 2006):

The VSH algorithm

1. Let $x = x_1 x_2 \ldots x_\ell$ be a message to be hashed with x_i the i^{th} bit of the message. Let the block length k be the largest integer for which $\prod_{i=1}^{k} p_i < N$. We assume $\ell < 2^k$.
2. Let $y_0 = 1$.
3. Let $d = \left\lceil \frac{\ell}{k} \right\rceil$ be the number of blocks. Let $x_i = 0$ for $\ell < i \leq dk$ be the padding.
4. Let $B = b_1 b_2 \ldots b_k$ be the binary representation of ℓ. Let $x_{dk+i} = b_i$ for $1 \leq i \leq k$.
5. For $j = 0, 1, \ldots, d$ in succession compute

$$y_{j+1} = y_j^2 \times \prod_{i=1}^{k} p_i^{x_{(jk+i)}} \mod N.$$

6. Return y_{d+1} as $H(x)$.

The authors of VSH claim that this algorithm offers security comparable to the security of RSA. By using some assumptions on the difficulty of factoring large integers, the complexity of the NFS algorithm and some approximation, the authors state that VSH with 1024-bit modulus achieves the same level of security as 840-bit RSA (Contini *et al.* 2006). This security claim is for the collision resistance of VSH, as can be seen from the next section. It should be noted that the VSH algorithm produces a family of hash functions with the modulus N acting as the index or key for the specific hash function from the family.

4.2 Collision resistance of the VSH

Now, we provide a proof for the collision resistance of VSH. The proof is a reduction from the collision finding problem to the (computational) NMSRVS assumption stated above.

Theorem 3. *Finding a collision for VSH is as hard as solving the NMSRVS problem (Contini* et al. *2006).*

Proof. Assume that we have two messages x and $x' \neq x$ with the same hash value under H, which is a VSH hash function. We denote by y_i the intermediate values in the computation of $H(x)$ and y'_j respectively for $H(x')$. We also define L and ℓ as the number of blocks in and the length of x. For x' the values L' and ℓ' are defined respectively. Now, because x and x' collide, we have $y_{L+1} = y'_{L'+1}$.

First, we consider the case where $\ell = \ell'$. We denote by x_i the i^{th} block in the message x and by b_j the j^{th} bit in the message x (x'_i and b'_j for x' respectively). Let $t \leq L$ be the largest index for which $(y_t, x_t) \neq (y'_t, x'_t)$ and $(y_i, x_i) = (y'_i, x'_i)$ for all $i, t < i \leq L$. Thus we have

$$(y_t)^2 \times \prod_{i=1}^{k} p_i^{b_{tk+i}} \equiv (y'_t)^2 \times \prod_{i=1}^{k} p_i^{b'_{tk+i}} \mod N. \tag{3}$$

Now, we define two sets $\Delta = \{i \mid b_{tk+i} \neq b'_{tk+i}, 1 \leq i \leq k\}$ and $\Gamma = \{i \mid b_{tk+i} = 1, b'_{tk+i} = 0, 1 \leq i \leq k\}$. Since all elements in (3) are invertible modulo N, we have

$$\Big(\frac{y_t}{y'_t} \times \prod_{i \in \Gamma} p_i\Big)^2 \equiv \prod_{i \in \Delta} p_i \mod N. \tag{4}$$

If $\Delta \neq \emptyset$, then (4) solves the NMSRVS problem. If $\Delta = \emptyset$, then $(y_t)^2 \equiv (y'_t)^2 \mod N$ and $t \geq 1$ by definition. If $y_t \not\equiv \pm y'_t \mod N$, then NMSRVS can be solved by factoring N. Now, if $y_t \equiv \pm y'_t \mod n$, then $\Delta = \emptyset$ and the definition of t guarantee that $y_t \equiv -y'_t \mod N$. Now, we consider the values y_{t-1} and y'_{t-1}. By proceeding as in (3) and in (4), we get that $(\frac{y_{t-1}}{y'_{t-1}})^2$ is equivalent to some very smooth number times $-1 \mod N$ and this solves the NMSRVS problem. Now, the case $\ell = \ell'$ is finished.

Finally, consider the case $\ell \neq \ell'$. We have $y_{L+1} = y'_{L'+1}$ and thus

$$\Big(\frac{y_L}{y'_{L'}}\Big)^2 \equiv \prod_{i=1}^{k} p_i^{\ell'_i - \ell_i} \mod N. \tag{5}$$

Furthermore, $|\ell'_i - \ell_i| = 1$ for some i and the NMSRVS problem can be solved with a similar transformation as in (4).

□

Although the above proof relates the collision resistance of VSH to the NMSRVS problem, there has been some research on the collision properties of VSH. Blake & Shparlinski (2007) provide an analysis of the statistical distribution of the VSH collisions. This leads to some novel number theoretic problems, but does not undermine the security against collisions provided by the above proof.

4.3 Preimage resistance of the VSH

Even the authors of the VSH hash function state that it does not provide preimage security for short messages when there is no wraparound modulo N (Contini *et al.* 2006). The reason why this is not considered too restricting is that in some protocols only collision resistance of the hash function is necessary (Menezes *et al.* 1996, p. 327) and that there is no proof of the existence of one-way functions (Contini *et al.* 2006). Thus, the lack of preimage resistance is not significant although for a general purpose hash function it would be essential.

However, Saarinen (2006) has found an interesting mathematical relation of VSH hash values and demonstrated that it could be used to find preimages of short passwords. In Publications I and II, we improved this method and demonstrated how to find preimages of multiple eight character passwords secured with VSH. In Publication VI, we demonstrated that similar attacks are applicable against the discrete logarithm variants of VSH.

4.3.1 Saarinen's method for finding preimages

Let H be a VSH hash function and x, y and z messages of equal length with $x \wedge y = z$ and $z = \vec{0}$. Now, Saarinen (2006) has shown that

$$H(x)H(y) \equiv H(x \vee y)H(z) \mod N. \tag{6}$$

The equivalence follows from the simple fact that the condition $x \wedge y = z$ ensures that the messages x and y do not have the same bits set to one. Thus, in the computation of the hash values $H(x)$ and $H(y)$, there are exactly the same primes with the same exponents as in $H(x \vee y)$. The value $H(z)$ is needed to balance the equivalence as the

block containing the length of the message gets calculated twice on the left hand side of (6).

With the help of the above result, Saarinen (2006) presents the following method for finding preimages of VSH hash values: Let x be the message for which the preimage is wanted. Thus, $H(x)$ is the hash value of x and this value is known to the attacker. Now, let ℓ be the length of the message x. Furthermore, x can be divided into two messages x_1 and x_2 of length ℓ by setting the $\frac{\ell}{2}$ first bits of x_1 equal to those in x and the rest equal to zero. Then, the first $\frac{\ell}{2}$ bits of x_2 are set to zero and the last bits equal to the corresponding bits of x. If the length of x is odd, then the odd bit can be taken into either of the halves. Clearly, the assumptions of (6) hold for x, x_1 and x_2. Thus, by (6) we have

$$H(x_1)H(x_2) \equiv H(x)H(z) \mod N.$$

Since the VSH hash values are invertible modulo N, we get

$$H(x_1) \equiv H(x)H(z)H(x_2)^{-1} \mod N. \tag{7}$$

Now, Saarinen (2006) shows that (7) can be used to find preimages of the hash values quite efficiently. One first tabulates the right hand side values of (7) for all possible values of x_2. Then, one starts to go through the left hand side values for all possible values of x_1. Once a match is found between the table and the left hand side values, the original message can be reconstructed with $x = x_1 \vee x_2$. Saarinen (2006) demonstrates this method with a toy example where he uses much smaller primes than suggested by the authors of VSH and lowercase alphabet passwords of only four characters.

4.3.2 Improvements and practical results

As the practical experiment made by Saarinen (2006) was not made with such security parameters as required by Contini *et al.* (2006), we decided to experiment with security parameters chosen to be large enough and with 8-character passwords, which would give a more realistic picture of the effectiveness of the method. Furthermore, we found ways to improve the method and to efficiently find preimages of large sets of passwords secured with the same instance of VSH.

First of all, we noticed that the order of tabulation and search in (7) could be changed, so that the left hand side is tabulated and the right hand side is used while searching for a match. This also facilitates the reuse of the generated table for different messages because the tabulated values do not depend on the original message x. However, this did

not give any advantage to the running time in finding the preimage of a single password, but it made it faster for subsequent passwords secured with the same implementation of VSH.

We tested four different versions of VSH: 1024-bit, 2048-bit and the cubing variants of these (Contini *et al.* 2006). In the cubing variant of VSH, the compression function in the fifth step of the algorithm is replaced with the following compression function

$$y_{j+1} = y_j^3 \times \prod_{i=1}^{k} p_i^{x_{(jk+i)}} \mod N.$$

We made our own implementations of the VSH hash functions, as it provided a level ground for testing all three methods for finding preimages. Programming was done with the Python programming language (Python Software Foundation 2007), which offered support for large number arithmetic. It should be noted that our implementation of VSH was not in any way optimised, so in an optimised environment, the time needed to find a preimage would be less than the times found in our research. The precomputed values were stored in a hash table for easy and fast access. In the searching phase, the possible prefixes were used in alphabetical order with lowercase alphabets first and numerical values last. The comparison was then done and there was some printing functionality to ensure that the correct password had been identified. The method worked 100% of the time.

The tests were performed with quite modest hardware as we had one 2.2 GHz processor and 8 GB of memory available. However, the memory needed for the computations was only 2-4 GB with the 1024-bit VSH and up to 9 GB for 2048-bit VSH. At least for the 1024-bit VSH, the amount of memory could easily be available in a modern desktop computer and thus these methods could be applied without any expensive special hardware or software.

The results for the different moduli and different variants of VSH are summarised in Tables 1 and 2. Table 1 presents the runtime of the algorithm divided between precomputation and the searching, where the best case and the worst case runtimes after precomputation with our method are given. Table 2 presents the best and worst case times with Saarinen's method. The precomputation time is not separated in this case as the method requires one to compute the tables for comparison for each message separately.

Table 1. Results for our method.

VSH type	Precomputation	Best case	Worst case
1024-bit	2.5 h	0 s	6.5 h
2048-bit	6 h	0 s	21 h
1024-bit cubing variant	2.5 h	0 s	10 h
2048-bit cubing variant	6 h	0 s	17 h

Table 2. Results for Saarinen's method.

VSH type	Best case	Worst case
1024-bit	7.5 h	9.5 h
2048-bit	12 h	23 h
1024-bit cubing variant	9.5 h	12 h
2048-bit cubing variant	18 h	25 h

With our method, the runtime of the preimage finding algorithm varied between 2.5 and 9 hours for passwords secured with 1024-bit VSH. The precomputation took 2.5 hours and the searching phase was finished immediately for the best case (password `aaaaaaaa`) and took 6.5 hours in the worst case (password `99999999`) This was of course due to the naive search in alphabetical order. With 2048-bit VSH, the precomputation time was 6 hours and in the worst case the table look-ups took 21 hours. The times for the cubing variants were very similar to the normal VSH implementations, which was somewhat surprising. With Saarinen's method, the runtimes were greater than those of our method in the best case and similar or slightly faster than those of our method in the worst case. This held for all the variants in our study, as can be seen in the tables.

With subsequent preimage searches, our method outperformed that of Saarinen's as the precomputation did not have to be done again. Further research showed that (7) could be improved by transforming it into

$$H(x_1)H(x)^{-1}H(z)^{-1} \equiv H(x_2)^{-1} \mod N. \tag{8}$$

With the help of the above congruence, we can speed up the search phase by making the computationally intensive inverting in the tabulation phase. It should be noted that the hash value of the message ($H(x)$) and the hash value of the empty message ($H(z)$) are both constants, which can be inverted once and then used in all subsequent computations. This allows those values to be used in the searching phase without increasing the runtime significantly.

We tested all three methods, the original by Saarinen and our two variants, again with 1024-bit VSH and alphanumeric passwords with eight characters. We generated the passwords randomly for three different sets. The sets contained 5, 10 and 50 passwords and the results are summarised in Tables 3, 4 and 5. In the tables, the average, fastest and slowest times refer to searching times after the precomputation as far as our methods are concerned. For Saarinen's method, these times refer to the total times as the precomputation is an integral part of each preimage search and thus cannot be considered separately.

It should be noted that our task of finding the preimages of all the passwords in the given set is more difficult than trying to find a preimage of a single password in some given set. In the latter scenario, one would only need to find a single match from a set of possibilities. Our methods and that of Saarinen could also be used in the latter scenario. With our method, the search for a single match could be done with a single table and all the hash values of the passwords could be compared against this table. With the method of Saarinen, one would require a different table for each password and thus the search would be more complicated. However, we did not explore this scenario in our research although this attack scenario is very plausible from a practical point of view. Usually, a single password is enough for an attacker to gain access to the system.

Table 3. Results for five passwords.

Method	Precomputation	Total time	Average time
Saarinen (2006)	N/A	46.00 h	9.25 h
Publication I	2.25 h	17.75 h	3.00 h
Publication II	5.00 h	10.75 h	1.25 h

Table 4. Results for ten passwords.

Method	Precomputation	Total time	Average time
Saarinen (2006)	N/A	62.50 h	6.25 h
Publication I	2.25 h	36.50 h	3.50 h
Publication II	5.00 h	18.25 h	1.25 h

Table 5. Results for fifty passwords.

Method	Precomputation	Total time	Average time	Fastest time	Slowest time
Saarinen (2006)	N/A	433.75 h	8.75 h	7.50 h	9.75 h
Publication I	2.25 h	171.00 h	3.50 h	0.12 h	6.50 h
Publication II	5.00 h	63.25 h	1.25 h	0.04 h	2.50 h

As mentioned earlier, the original method of Saarinen does not have reusable tables and thus it is by far the most time consuming of the three methods tested for larger password sets. Since the precomputation is an integral part of finding each preimage, it is not mentioned for this method in the tables. With our first method, the tables are reusable and the precomputation phase is faster than with our second method. However, each of the searches within the precomputed tables took on the average almost three times as much time as with our second method, as Table 5 shows. Thus, with multiple passwords, the second method was much faster on the whole than either Saarinen's method or our previous method.

It is also worth noting that the precomputation of the tables with our two methods takes a constant time for a given implementation of VSH and a given alphabet for passwords and does not depend on the number of passwords to be searched. It can also be stored for later use if needed.

The better efficiency of our second method can be accounted for by the following observation: In the method from II, one needs to compute only one multiplication in the search phase. In the method from I there is one inversion, which is more expensive computationally when working with integers modulo N with our implementation. The Montgomery arithmetic by Montgomery (1985) could be utilised to exchange computationally intensive inversions with some extra multiplications. There is a method based on Montgomery arithmetic that inverts t elements with $3(t-1)$ multiplications and a single inversion (see for example the paper by Mishra & Sarkar (2003) for details).

The original method is not easily parallelisable, but there is a parallel version of this algorithm by Mishra & Sarkar (2003). However, the use of these algorithms would require some extra memory to accommodate for the intermediate results. This could be used to make a trade-off between the required extra memory and the speed up gained from the algorithms.

In the case where inverting an element is easy and the multiplication of elements is hard, the method from I could be more efficient in finding preimages of multiple hash values. An example of such a case would be elliptic curves, where the addition of elements is computationally much more intensive than inverting an element. Elliptic curves and VSH are discussed in more detail in Section 4.4.

Unfortunately, we have no formal model for predicting the runtime of our algorithms as a function of the hash length. Saarinen (2006) states that the complexity of his method is dependent only on the size of the alphabet over which the passwords are formed and is of the order $O(2^{\frac{k}{2}})$, where k is the size of the alphabet for the passwords. This can also be seen in our case. In the tabulation phase, we need to compute $2^{\frac{k}{2}}$ hash values and save these in the table. In the searching phase, the worst case is that we need to go through all the $2^{\frac{k}{2}}$ different values before a match is found. Thus, the overall complexity for finding a preimage is in $O(2^{\frac{k}{2}+1})$ when we take into account the possibility for an odd k. A normal brute force attack for finding a preimage in this alphabet would have complexity in $O(2^k)$.

If we consider (6) and its variations derived above, it can be easily seen that a useful congruence equation should have the two "halves" of the message on different sides of the congruence. Otherwise, the congruence cannot be applied in preimage searches. For the most efficient implementation against the basic VSH, one should have the inversion of the variable messages only during the tabulation phase. This would keep the computationally intensive use of the extended Euclid's algorithm in the tabulation phase and make the searches much faster. Moving the constant terms around does not really affect the computation time. Thus, the methods presented by Saarinen (2006) and in Publications I and II are the only variations of (6), which are usable in preimage attacks against VSH.

There is also an improvement to the running time of VSH by Bellare & Ristov (2008). This improvement would also further decrease the time needed to find preimages. However, it is only an improvement to the running time and thus does not protect against the preimage finding methods described earlier. An even more effective improvement would be a dictionary search on the passwords. In a dictionary search, one searches the

most probable passwords first and not just in alphabetical order. In any attack against a database of real (not random), user-generated passwords, a dictionary search would be more effective and result in even shorter times in the searching phase. This would not affect the precomputation time, unless only the most probable passwords were included in the table.

Our methods are also highly parallelisable and the computation can be divided among several processors with access to the precomputed table. Even the table can be divided into parts if more processors are available. The division can be done in a straightforward manner by simply taking (possibly equal size) parts of the table and making the matching on these tables with separate processors or computers altogether. This would lead to a far more effective method, especially if combined with a dictionary search on the passwords.

The practical results show that passwords secured with VSH can be found quite efficiently even for long moduli and for different variations of VSH. Also, the reusability of the tables makes it very fast to find several passwords secured with the same implementation of VSH. Fortunately, to the best of our knowledge, VSH has not been used in password protection in real systems. In fact, we are not aware of any real implementation of VSH that is used as a hash function in a cryptographic protocol.

4.4 Discrete logarithm variants of the VSH

The VSH hash function has quite many different variants as mentioned above and one of these is based on the discrete logarithm problem (VSH-DL). This was proposed by Contini *et al.* (2006) together with the original VSH hash function. The main idea is to change the hardness of collision finding from integer factoring to discrete logarithm in $\mathbb{Z}_p$ for a suitable large prime p.

Let $p = 2q+1$ be an s-bit prime with q a prime. Furthermore, let $k \in \mathbb{N}_+$ with $k \approx \frac{s}{\log s}$. Let $x = x_1 x_2 \cdots x_\ell$ be a binary message of length ℓ with $\ell < (s-2)k$. The VSH-DL compression function algorithm is the following:

The VSH-DL algorithm

1. Let $y_0 = 1$.
2. Let $d = \lceil \frac{\ell}{k} \rceil$ be the number of blocks. Let $x_i = 0$ for $\ell < i \leq dk$ be the padding.
3. Let $B = b_1 b_2 \ldots b_k$ be the binary representation of ℓ. Let $x_{dk+i} = b_i$ for $1 \leq i \leq k$.

4. For $j = 0, 1, \ldots, d$ in succession compute

$$y_{j+1} = y_j^2 \times \prod_{i=1}^{k} p_i^{x_{(jk+i)}} \mod p.$$

5. Return y_{d+1}.

This idea was further extended by Lenstra *et al.* (2006) in their paper, where they demonstrate that VSH-DL can be realised using the discrete logarithm problem in finite fields or elliptic curves. These modifications make the resulting hash value shorter, but the computation is somewhat slower.

The finite field variant of VSH-DL is based on the following assumptions and observations (Lenstra *et al.* 2006): First of all, recall the definitions of the DLP and the k-modified DLP from Chapter 3. The idea of the VSH-DL in finite fields is to use the efficient computation in these fields. Thus, one chooses p a prime and generates the sixth degree extension $\mathbb{F}_{p^6}$ of $\mathbb{F}_p$. Now, $|\mathbb{F}_{p^6}^*| = p^6 - 1$, which factors into $(p^2 - p + 1)(p^2 + p + 1)(p + 1)(p - 1)$. Thus, there is a unique subgroup of order $p^2 - p + 1$ in $\mathbb{F}_{p^6}^*$. This subgroup is claimed to contain the difficulty of the DLP for $\mathbb{F}_{p^6}^*$ as the other subgroups have some subexponential algorithms for solving the DLP (Lenstra 1997). We denote this subgroup of order $p^2 - p + 1$ by G in the following.

Now, Lenstra & Verheul (2000) and Rubin & Silverberg (2003) present methods for efficient computation in and compression on the subgroups of multiplicative groups of finite fields (XTR and CEILIDH). Thus, one does not necessarily need to use the arithmetic of the full field, but could compute in the subgroups instead. This makes the VSH-DL more efficient in finite fields. Unfortunately, these methods do not directly help in the arithmetic as CEILIDH is only a compression method and XTR is not applicable in the multiexponentiation of VSH. Thus, these methods can only be applied to compress the hash values to the subgroup and thus shorten the hash value.

The modified VSH-DL compression function is the following:

The modified VSH-DL compression function

1. Let D be a finite cyclic group of known and factored order and let d be a generator of D. Let K be a subgroup of D with a generator $g = d^{\frac{|D|}{|K|}}$. Let p be a prime dividing $|K|$ but not $|D|/p$ and $w = \lfloor \log_2 p \rfloor$. Let C be an efficiently computable injection and $k \in \mathbb{Z}_+$ such that $(w-1)k < 2^k$. Furthermore, let $\Psi : H \to H$ be a mapping for which $\Psi(d_i)^{\frac{|D|}{|K|}} = (d_i)^{\frac{|D|}{|K|}}$ for all $y_i \in D$. For $i = 1, 2, \ldots, k$ draw d_i uniformly at random from D.

2. Let $x = x_1 x_2 \cdots x_t$ be the message in binary, with $t \leq (w-1)k$ and set $y_0 = 1$.
3. Let $\ell = \lceil \frac{t}{k} \rceil$ be the number of message blocks in x. Pad the message by setting $x_i = 0$ for all $t < i \leq \ell k$.
4. Let $t = b_1 b_2 \cdots b_k$ be t in binary. Let $x_{\ell k+i} = b_i$ for all $1 \leq i \leq k$.
5. For $j = 0, 1, \ldots, \ell$ calculate

$$y_{j+1} = y_j^2 \times \prod_{i=1}^{k} \Psi(d_i)^{x_{(jk+i)}}.$$

6. Return $C(y_{\ell+1}^{\frac{|D|}{|K|}})$ as the hash value of x.

Lenstra *et al.* (2006) state that this function should be instantiated with $D = \mathbb{F}_{p^6}^*$, $K = G$ and compression should be done with CEILIDH. These choices would facilitate relatively fast computation and shorter hash values.

The elliptic curve variant of VSH-DL modifies the original only in the sense that instead of multiplying primes in the compression phase, we use points on the elliptic curve and addition instead of multiplication. Thus, the compression step in the algorithm would be

$$y_{i+1} = 2 \times y_i + \sum_{j=1}^{k} P_j,$$

where P_j are some points on the elliptic curve.

In VI, we have demonstrated that these variants are susceptible to similar preimage finding attacks as the basic VSH (and its cubing variant). These results are based on the following observations:

First of all, it should be noted that the VSH-DL compression function is invertible unlike the basic VSH. Thus, the meet-in-the-middle attack presented by Lai & Massey (1993) can be directly applied. This attack is a generic one, that works for all invertible compression functions and does not take into account any special features of VSH. It is also a probabilistic attack that works with a very high probability, but is not guaranteed to succeed with a single run or against a specific password from the alphabet. In addition, finding a preimage of a single block message (without the length block) is trivial for VSH-DL.

When the previous methods presented against VSH are considered against VSH-DL, it is easy to see that (6) applies to the original VSH-DL compression function. Exactly the same arguments as with the basic VSH can be applied. Thus, all the methods presented in the previous sections apply directly to the VSH-DL.

Now the first interesting case is the modified VSH-DL compression function. From VI, we have a result, which shows that similar equation as (6) can be devised for the modified VSH-DL as well.

Theorem 4. *Let x, y and z be messages of equal length, with $x \wedge y = \vec{0} = z$. Let H be a modified VSH-DL compression function. Then, the following equation holds*

$$H(x)H(y) = H(x \vee y)H(z). \tag{9}$$

The above result follows from the commutativity of the multiplication, which allows us to sum up the exponents on both sides of (9), and show that these sums are equal. The compression in the end can be reversed and the application of the mapping Ψ can be ignored as the exponent $\frac{|D|}{|K|}$ is used in the end.

It is, of course, evident that because also the elliptic curve computations are commutative, we have the same result for these as well. The only difference is again the change from multiplication to addition, but this is irrelevant to the result of Theorem 4.

Another generalisation we prove in VI is the following:

Theorem 5. *Let $k \in \mathbb{N}_+$ and $x_1, x_2, \ldots x_k, z$ be messages of equal length with $x_i \wedge x_j = \vec{0} = z$ for all $i, j \in \{1, 2, \ldots, k\}, i \neq j$. Furthermore, let H be a VSH or VSH-DL hash function. Then the following equivalence holds:*

$$H(x_1)H(x_2)\cdots H(x_k) \equiv H(x_1 \vee x_2 \vee \cdots \vee x_k)H(z)^{k-1} \mod N. \tag{10}$$

This result is a simple induction over k with (6) as the basis step. Theorem 5 shows, that by gaining information on the message, we may utilise that information to fix parts of the message and only hash the unknown parts and then combine these to get the full hash of the message. Also the computation of the hash values of the distinct parts of the message should be quite fast as there will be a lot of zero bits in these messages. Of course, even with other hash functions, one can fix the known parts and then only vary the variable parts of the message. However, with many other hash functions, one needs to compute the full hash function for all the messages.

With the help of the above results, we propose in VI a general method for preimage finding for VSH, VSH-DL and the variants discussed in this thesis.

A general preimage finding algorithm for the VSH and the VSH-DL variants

1. Let $H(x)$ be a hash value for a VSH or VSH-DL variant H. Let x be a (partially) unknown message of known length of ℓ bits. Let $t \in \mathbb{N}$ be the number of distinct known parts of x and z a zero vector of length ℓ.

2. Now, divide x into $x = v_0 w_1 v_1 w_2 \cdots v_{t-1} w_t v_t$, where each $w_i, i \in \{1,2,\ldots,t\}$ is a known part of x and each $v_j, j \in \{0,1,\ldots,t\}$ is an unknown part of x.
3. Let w_i' be a message of length ℓ by setting all other parts of x except w_i to zeroes for all $i \in \{1,2,\ldots,t\}$. Similarly, form message variables v_j' (of length ℓ) from the unknown parts v_j for all $j \in \{0,1,\ldots,t\}$.
4. Divide the possible preimage space into two parts by forming the equation

$$\prod_{i=1}^{t} H(w_i') \times \prod_{j=0}^{\lceil \frac{t}{2} \rceil} H(v_j') = H(z)^{2t} H(m) \times \prod_{j=\lceil \frac{t}{2} \rceil + 1}^{t} H(v_j')^{-1} \tag{11}$$

   with the help of Theorem 5.
5. Compute and tabulate the values of the left hand side of (11) for all possible values of all the message variables v_j' on the left hand side.
6. Compute the right hand side of (11) for all possible values of all the message variables v_j' on the right hand side until a match between the tabulated values is found.
7. When a match is found between the table and the searched values, $x = v_0' \vee w_1' \vee v_1' \vee \cdots \vee v_{t-1}' \vee w_t' \vee v_t'$ for the values of v_j', which yield the matching between the table and the search.
8. Return the preimage x of $H(x)$.

As the preimage finding methods presented by Saarinen and those in this thesis are all meet-in-the-middle attacks, these could benefit from the methods presented by Quisquater & Delescaille (1990) and van Oorschot & Wiener (1996). These would enable a memoryless way of performing the preimage searches. However, these would not be directly applicable in our scenario, where several hash values of passwords need to be inverted.

One should also notice that adding a salting value to the computation of the hash value of the password will not be effective, if the attacker knows the length of the salt and the place in which it is input in the computation of the hash. The attacker can then take this information into account when dividing the preimage space and constructing the table leaving the part where the salt is applied into the searching phase. When the attacker learns the salt and the hash value of the password, the attacker can use the above algorithm and the salt transforms into a constant in the search with the other parts of the message varying through the preimage space.

5 Multicollisions and iterated hash functions

In this chapter, we define the concept of multicollisions for hash functions as a natural extension to collisions. Then, we present the most significant findings on multicollisions for iterated hash functions and discuss the practical implications. We go on to describe some countermeasures against the multicollision attack discovered by Joux (2004) and discuss the effectiveness of these countermeasures. We also discuss the generalised iterated hash functions and their security against multicollisions. In our research, we have developed a theoretical framework for studying (generalised) iterated hash functions. We present the findings of this research without many of the detailed proofs.

5.1 Multicollisions on iterated hash functions

Next, we define the concept of multicollisions as the generalisations of collisions. Let $k \in \mathbb{N}_+$. A *k-collision* on the hash function $h : \{0,1\}^* \to \{0,1\}^n$ is a set $A \subseteq \{0,1\}^*$ such that $|A| = k$ and $h(x) = h(y)$ for all $x, y \in A$. A collision (on h) is thus a 2-collision on h.

A k-collision attack (algorithm) on a hash function h can loosely be characterised to be a probabilistic process that finds a k-collision on h with some non-negligible probability. According to the (generalised) birthday paradox, a k-collision can be found (with probability approx. $\frac{1}{2}$) by hashing $(k!)^{\frac{1}{k}} 2^{\frac{n(k-1)}{k}}$ messages (see for example Girault & Stern (1994) and Suzuki *et al.* (2008)). In the case $k = 2$, this gives $\sqrt{2} \cdot 2^{\frac{n}{2}}$, which is in $O(2^{\frac{n}{2}})$.

As mentioned in Sections 3.4 and 3.5, finding collisions could be done in $O(2^{\frac{n}{2}})$ time complexity. Also, finding a k-collision for an arbitrary hash function requires hashing approximately $\Omega((k!)^{\frac{1}{k}} 2^{\frac{n(k-1)}{k}})$ messages. However, for iterated hash functions and generalised iterated hash functions, finding multicollisions is much easier than that. First, we take a look at the famous multicollision attack by Joux (2004).

In his paper, Joux (2004) describes a novel method for generating multicollisions for iterated hash functions. The method is quite simple and ingenious. By generating successive collisions to the underlying compression function, one can form a 2^k-collision with only k collision attacks, i.e. with a complexity in $O(k2^{\frac{n}{2}})$.

Formally, the method of Joux (2004) works as follows: Let h be an iterated hash function with f as the underlying compression function and let y_0 be the initial value of h. Now, we use the birthday attack to find two message blocks x_1, x_1' with $x_1 \neq x_1'$ such that $f(y_0, x_1) = f(y_0, x_1') = y_1$. After this, we can perform the birthday attack on f with the initial value y_1 to find message blocks x_2 and x_2' for which $f(y_1, x_2) = f(y_1, x_2') = y_2$. After k such steps we have k pairs of messages out of which we can form 2^k distinct messages with the same hash value. Note that all the messages have equal length, so even MD-strengthening does not protect against this type of attack. Thus, the multicollision attack on the iterated compression function extends to the full hash function.

Joux (2004) uses this new method to show that the folklore assumptions on the security of *catenated hash functions* are false. Catenated hash functions are formed from two or more different hash functions. One computes the hash value of the message with all of these and then catenates these results to form the final hash value. Joux (2004) is able to show, that the security of this type of construction against preimage and collision attacks is much weaker than previously assumed, when the catenation is done with two hash functions and one of the hash functions is an iterated hash function. Let h and h' be hash functions of length n_1 and n_2, respectively. Now, if catenated hash functions were ideal, the complexity for finding collisions would be in $O(2^{\frac{n_1+n_2}{2}})$. Joux (2004) proves with the help of the multicollision technique presented above, that the complexity is in $O(n_2 2^{\frac{n_1}{2}} + 2^{\frac{n_2}{2}})$, when $n_2 \geq n_1$. If $n_1 \geq n_2$, then their roles in the complexity can be exchanged. Also, finding (second) preimages should be in $O(2^{n_1+n_2})$, but by Joux's results it is in $O(n_1 2^{\frac{n_2}{2}} + 2^{n_2} + 2^{n_1})$, if $n_2 \geq n_1$.

With some added assumptions, Kelsey & Schneier (2005) and Aumasson (2008) have shown that arbitrarily large multicollisions can actually be found in almost $O(2^{\frac{n}{2}})$ time. The method of Kelsey & Schneier (2005) is based on the idea of *expandable messages*. Let $a, b \in \mathbb{N}_+$ be such that $a < b$, $y_0 \in \{0,1\}^n$ an initial value and f a compression function of length n. Now, an (a,b)-expandable message is a set of messages $\{x_a, x_{a+1}, \ldots, x_b\}$, where the length of the message x_i is equal to i blocks for all $i \in \{a, a+1, \ldots, b\}$ and $f^*(y_0, x_j) = f^*(y_0, x_\ell)$ for all $j, \ell \in \{a, a+1, \ldots, b\}$. Kelsey & Schneier (2005) give a method for generating $(t, t+2^t-1)$-expandable messages with complexity in $O(t 2^{\frac{n}{2}+1})$, where $t \in \mathbb{N}_+$. With the help of these expandable messages, one can generate multicollisions in different ways.

A naive method with a pair of (a,b)-expandable messages allows one to find $b-a+1$ different messages with the same hash value, namely by catenating the two messages together with $a+i$ blocks on the first and $b-i$ blocks on the second message

for $0 \leq i \leq b-a$. As the complexity of constructing expandable messages is fairly small, one can make more and more expandable messages of the same size and then combine these in the same way as before with only two expandable messages. Thus, by generating s different (a,b)-expandable messages one can form $\binom{b-a+1}{s}$-collisions when increasing the work only by a factor of s. Thus, huge collisions can be generated with complexity in $O(t2^{\frac{n}{2}+1})$. Kelsey & Schneier (2005) use these expandable messages also to generate second preimages for very long messages.

The multicollision finding method of Aumasson (2008) is based on the concept of *fixed points* of the underlying compression function. A fixed point of a compression function f is a pair (y,x), where $y \in \{0,1\}^n$ and $x \in \{0,1\}^m$, such that $f(y,x) = y$. Fixed points have been studied already by Dean (1999) in the context of second preimage attacks. The idea behind Aumasson's method is that one finds a pair of fixed points with colliding hash values, i.e. fixed points (y,x) and (y,x') with $x \neq x'$ instead of a colliding pair of (random) messages. If fixed points could be found very fast, i.e. assume that finding a fixed point takes unit time, the complexity of finding arbitrarily large multicollisions is essentially in $O(2^{\frac{n}{2}})$. The method works as follows: Let $k \in \mathbb{N}_+$. Generate a fixed point collision (y_0,x) and (y_0,x') for the initial value y_0 of the given hash function. Now, all the messages from $\{x,x'\}^k$ form a 2^k-collision. Moreover, the complexity of generating this is the complexity of finding a fixed point collision, i.e. in $O(2^{\frac{n}{2}})$, if we assume that finding a single fixed point takes unit time. Also Kelsey & Schneier (2005) demonstrate that their method can be used with fixed points, but it is not as efficient and generates longer messages if used in multicollisions.

Fixed points are very useful in constructing second preimages especially for long messages. Several methods for constructing second preimages with the help of fixed points have been presented in the thesis of Dean (Dean 1999, pp. 120-125). Also some of the limitations of these methods are discussed there. Finding second preimages of long messages is also the main application of the multicollision finding method of Kelsey & Schneier (2005).

In our research, we have found a method for finding arbitrarily large multicollisions for iterated hash functions with the complexity in $O(2^{\frac{n}{2}})$ for any compression function, even a FIL-RO. This idea is presented in VI. The idea is very simple and is a generalisation of the ideas of Dean (1999), Kelsey & Schneier (2005) and Aumasson (2008), but it has some serious drawbacks, which make it impractical.

Our method works in the following way: Let $y_0 \in \{0,1\}^n, x_1 \in \{0,1\}^m$ and f a compression function of length n and block length m. Compute the values $f^*(y_0,x_1^i)$

for $i = 1,2,\ldots,2^{\frac{n}{2}}$. By the results of Flajolet & Odlyzko (1990), there are (with high probability) two values $l,k \in \{1,2,\ldots,2^{\frac{n}{2}}\}, l \neq k$, for which $f^*(y_0,x_1^k) = f^*(y_0,x_1^l) = y_0'$. Without loss of generality, we may assume $k < l$ and define $s = l - k$. Now, take $x_2 \in \{0,1\}^n, x_2 \neq x_1$ and compute the values $f^*(y_0',x_2^j)$ for $j = 1,2,\ldots,2^{\frac{n}{2}}$. Again, there are (with high probability) two values $c,d \in \{1,2,\ldots,2^{\frac{n}{2}}\}, c \neq d$, for which $f^*(y_0',x_2^c) = f^*(y_0',x_2^d) = y_1$ and we may assume $c < d$ and define $r = d - c$.

Now, for any $t \in \mathbb{N}_+$, we may use the two cycles in the iteration to form a t-collision for the iterated hash function using f as a compression function. Let g be the least common multiplier of r and s. Now, take the set of messages $C_t = \{x_1^k x_1^{gi} x_2^c x_2^{g(t-1-i)} \mid i = 0,1,\ldots,t-1\}$. The messages are all distinct, because $x_2 \neq x_1$. Figure 2 illustrates the generation of the set C_t. It should be noted that all the messages in C_t have the same length and thus MD-strengthening does not protect against this attack. Thus, C_t forms a t-collision for the iterated hash function based on f. The complexity of this attack is the same as finding two collisions, i.e. in $O(2^{\frac{n}{2}})$. However, the length of the messages generated in this way is in $O(2^n)$ message blocks, which is completely impractical.

Fig 2. Our multicollision method with computational complexity in $O(2^{\frac{n}{2}})$ and message length in $O(2^n)$.

Our method can be applied together with Joux's method and without increasing the asymptotic complexity, we can form arbitrarily large multicollisions with the number of message blocks in $O(2^{\frac{n}{2}})$. Let y_0 and x_1 be as above. We first begin as in the previous method by computing $f^*(y_0,x_1^i) = y_i$ for $i = 1,2,\ldots,2^{\frac{n}{2}}$. Then, we use Joux's method to find message blocks z_1 and z_2 such that $z_1 \neq z_2$ and $f(y_0',z_1) = f(y_0',z_2) = u_0$. Then, we proceed by finding a cycle spanning over the message blocks y and z, i.e. $f^*(u_0,x_1^p) = u_p = y_j$ for some $p,j \in \{1,2,\ldots,2^{\frac{n}{2}}\}$. After this, we can form the set

$$D_q = \left\{x_1^j \{x_1^{2^{\frac{n}{2}}-j} z_1 x_1^p, x_1^{2^{\frac{n}{2}}-j} z_2 x_1^p\}^q\right\},$$

where $q \in \mathbb{N}_+$. Now, D_q is a 2^q-collision for the iterated compression function f^*. The length of the messages in the set D_q is in $O(q2^{\frac{n}{2}})$, which makes the messages still too long for any practical attack against real iterated hash functions.

In IV, there is also a method, which generalises Joux's attack in a way, that yields larger multicollisions for the same amount of work, if there is abundant memory available for the attacker and the memory access rate is not too high (Wiener 2004). The exact exposition of this method can be found in IV and possibly in a future thesis by M.Sc. Tuomas Kortelainen and thus only the main results without proofs are given here.

In this new method, we generate smaller sets of message blocks and find collisions between these small sets. The lengths of these messages are not all equal, but with some approximation, we are able to show in IV that this method yields after k steps a collision of size approximately $(e^{\sqrt{2}})^k$. This limit is reached when the amount of memory approaches infinity, but even with restricted memory, we are able to show that our method outperforms Joux's attack, when both utilise the same amount of calls to the underlying compression function (see IV for details).

Based on these theoretical results and the results of Joux (2004), we can form a theorem concerning the limits of iterated, MD-type hash functions.

Theorem 6. *Let h be a hash function of the MD-type with a FIL-RO f as its compression function. Finding arbitrarily large sets of preimages, second preimages or collisions for h has at most the same asymptotic complexity as finding a single preimage, second preimage or collision, respectively.*

Proof. The statements for preimages and second preimages follow from the results of Joux (2004) and the fact that any multicollision can be made into a preimage or second preimage by adding a single preimage attack from the common hash value of the multicollision in the end. The result for collisions is a direct consequence from our attacks demonstrated above. □

It should be noted that the above theorem does not take into account the complexity of writing down the explicit messages in these sets. Especially with collisions, these messages have lengths in $O(2^n)$ and thus writing down the messages would have greater complexity than the attack to find them. However, these messages can be stored in memory by only recording the message blocks and the lengths of the cycles that can be used to produce the multicollisions of desired sizes.

5.2 Countermeasures against Joux's multicollision attack

After Joux (2004) had published his multicollision finding method, there were several attempts to enhance the iterative construction of hash functions, especially against the multicollision finding method. Next, we present some of the most widespread ideas. Some of these methods have also been broken, whereas others, especially the wide pipe construction by Lucks (2005), have had more success and are now used in many of the proposals for the SHA-3 standard. This is by no means an exhaustive list of all the methods of improvement proposed either in general or as a part of some concrete hash function proposal.

5.2.1 Wide pipe and double pipe hashing

Lucks (2005) presented an analysis of two different methods, which work against Joux's multicollision finding method. These methods are called *wide pipe hashing* and *double pipe hashing*. The methods give better security against multicollisions than the basic MD-type of iteration, but come with some computational costs.

In the wide-pipe hashing method, one devises two compression functions f and g, with $f : \{0,1\}^l \to \{0,1\}^m$ and $g : \{0,1\}^m \to \{0,1\}^n$, where $l > m > n$. Then, one constructs an iterated hash function from f in the traditional manner. From the final output of this hash function, one further compresses the result by using g. This is then the hash value of the message. An early example of a wide-pipe design is the MDC-2 hash function by Coppersmith *et al.* (1990).

The improved security of the wide pipe hash is based on the following observation. If $m \geq 2n$, then Joux's attack on the iteration part of the wide pipe hash has complexity $O(k2^{\frac{m}{2}})$. Now, $m \geq 2n$ guarantees that this is greater than $O(k2^n)$. Then again, multicollisions for any hash or compression function can be found with complexity $O((k!)^{\frac{1}{k}} 2^{\frac{n(k-1)}{k}})$ as noted earlier. Since g is not involved in the iteration phase, a multicollision attack for g would have the complexity $O((k!)^{\frac{1}{k}} 2^{\frac{n(k-1)}{k}})$, assuming that g is a FIL-RO. Thus, Joux's method does not improve the multicollision or (second) preimage attacks against these hash functions, when the final hash length is considered as the security parameter.

Furthermore, the results of Coron *et al.* (2005) show that by using two independent FIL-ROs in the manner of the wide pipe construction, one achieves indifferentiability

from a random oracle. Thus, Lucks' wide pipe hash brings added security to the basic iterative construction. There is some computational and memory overhead in using two distinct compression functions and having the iterative structure containing a much larger state than in the MD-type of iteration. This seems to be a reasonable trade-off in practice as the wide pipe construction has been used in some of the proposals for the SHA-3 competition. Together with HAIFA (Biham & Dunkelman 2007), it seems to be a popular choice for improving the security of iterated hash functions against multicollisions. The description and analysis of HAIFA are out of the scope of this thesis and the reader is encouraged to read the original article by Biham & Dunkelman (2007) for a full description.

The other type of countermeasure against multicollision attacks analysed by Lucks (2005) is the double pipe hash. The double pipe hash uses one compression function f with three different initial values and interaction between the two "pipes" as follows: Let f be a compression function with length n and block length $n+m, m \geq n$ and y_0, v_0 and w three distinct initial values from $\{0,1\}^n$. Now we set $y_i = f(y_{i-1}, v_{i-1}m_i)$ and $v_i = f(v_{i-1}, y_{i-1}m_i)$ for all $i = 1,2,\ldots,\ell$, where ℓ is the number of blocks in the message to be hashed and m_i denotes the i^{th} block of the message. The final hash value is defined as $y_{\text{fin}} = f(w, y_\ell v_\ell 0^{m-n})$. Lucks (2005) is able to prove that also this construction improves the security of the basic iteration greatly as did the wide pipe hash. However, the drawback is that it doubles the number of calls to the underlying compression function. Thus, this type of construction has not achieved such widespread use as the wide pipe construction.

5.2.2 Checksums

Some iterated constructions employ checksums to ward against the multicollision attack proposed by Joux. For example, the 3C-construction of Gauravaram *et al.* (2006) uses a simple XOR-sum, which is carried through the iteration and updated after every iteration step together with the intermediate hash values. In the final step of the procedure, this sum is then XORed to the final value of the iteration to obtain the final hash value. This way, the basic attack of Joux could not work as the attacker is required to find colliding messages with proper XOR-sums. Also the GOST hash function uses this type of checksum method as an added security feature although the checksum is calculated with modular addition instead of XOR.

Unfortunately, there are multicollision attacks against these types of hash functions for example by Gauravaram & Kelsey (2007, 2008). Thus, the method has lost its appeal as a security measure against multicollision attacks.

5.2.3 Dithering

Rivest (2005) proposes to add some dither information to hash calculations. This information would be added to each message block and if this added information would not have any discernible patterns, then the attack of Joux would not work in its basic form. Rivest proposes several methods for dithering, but also finds many of them too simple to offer any real protection against multicollisions.

The one method Rivest finds feasible and secure enough is dithering with the help of Abelian square-free words. These words have very aperiodic nature as they do not contain any Abelian squares as a subword, i.e. a word and its permutation catenated. Thus, these words could be used as the dithering information added to each message block. Due to results of Keränen (1992), there is an efficient method for generating Abelian square-free words in four letters. Rivest proposes this as the method for generating the dither data.

There are two concrete proposals by Rivest (2005). First of all, one could just use two bits to code the letters of the four-letter alphabet (00, 01, 10 and 11) and catenate the letters from the Keränen's word to the end of each message block. This way, there would be minimal overhead as only two bits would be required for each message block. The other proposal of Rivest is to use sixteen bits for each dithering symbol. In this method, each dithering symbol except for the last block would be of the form 0 catenated with the $\lfloor \frac{i}{2^{13}} \rfloor^{\text{th}}$ symbol of Keränen's sequence and the binary representation of $i \mod 2^{13}$. The final block would have as its dithering input 1 catenated with the length of the message (in blocks). This way, the Keränen's sequence only has to be updated very rarely and the generation of the dithering input becomes much faster.

There are second preimage attacks against dithered hash functions by Andreeva *et al.* (2008) and thus this approach does not in general provide much added security against multicollisions and second preimages.

5.2.4 Generalised iterated hash functions

The idea of using message blocks more than once has been used in specific constructions in message authentication already by Davies & Price (1980). This construction was found insecure by Coppersmith (1986). Allowing the message blocks also to be used in a permuted order in the computation of the hash function, leads into the notion of *generalised iterated hash functions* that have been analysed by Nandi & Stinson (2007), Hoch & Shamir (2006) and in Publications III, V and VII of this thesis.

Nandi & Stinson (2007) define a very general class of (iterated) hash functions informally as follows. Let f be a compression function. A hash function H from the class behaves in the following way:

1. It invokes f a finite number of times.
2. The entire output of any intermediate invocation (not the final invocation) is input to some subsequent invocation of f.
3. Each bit of the message to be hashed is input into at least one invocation of f.
4. The output of the final invocation is the output of the hash function H.

More formally, Nandi & Stinson (2007) define a class of hash functions $\mathscr{D}$, which fulfil the above criteria. Let f be a compression function of length n and block length m. We define a set $V = \{v_1, v_2, \ldots\}$ of initial values (of length n) and the message $m = m_1 m_2 \cdots m_t$, where the length of each message block is k. Now, at each iteration, the input of f is formed from d different initial values and previous hash values and d' different message blocks by catenating these values. Now, we must have $dn + d'k = n + m$ for f to be able to process this combined variable. The properties of hash functions in this very general class of hash functions have not been studied extensively. Some preliminary results have been given by Nandi & Stinson (2004), Hoch & Shamir (2006) and the author of this thesis in VII.

Nandi & Stinson (2007) define a subclass of $\mathscr{D}$, which they call generalised sequential hash functions. In our terminology, these are called *generalised iterated hash functions*, which will be defined formally in the following section. The results of Nandi & Stinson (2007) show that using message blocks at most twice in a generalised iterated hash function does not improve the security against multicollisions as much as could be expected. Their result states, that the complexity of finding a 2^r-collision for a generalised iterated hash function with the above restriction has the average case complexity in $O(r^2 \cdot (\ln r) \cdot (n + \ln(\ln 2r)) \cdot 2^{\frac{n}{2}})$.

Hoch & Shamir (2006) have generalised the results of Nandi & Stinson (2007) and argue that even if we allow message blocks to be used $q \in \mathbb{N}_+$ times, multicollisions can be found for generalised iterated hash functions. In our research, we were able to improve the analysis of these complexity bounds. We outline our results in the next section. Hoch & Shamir (2006) also define slightly more general subclass of $\mathscr{D}$, so-called *tree based hash functions*, and show that their results apply for binary tree-based hash functions. This work has been extended in VII and will be discussed in more detail in Section 5.4.

5.3 The Nested Multicollision Attack Schema

In the following, we shall define the concept of a generalised iterated hash function that was mentioned above. We assume that all messages are in a block representation form. By a block representation, we mean the padding and division of the message into full blocks of equal length.

Let $l \in \mathbb{N}_+$ and $\alpha \in \mathbb{N}_l^*$ be such that $\alpha = a_1 a_2 \cdots a_s$, where $s \in \mathbb{N}$ and $a_i \in \mathbb{N}_l$ for $i = 1, 2, \ldots, s$. Note that here $\mathbb{N}_l^*$ is the free monoid generated by the alphabet $\mathbb{N}_l = \{1, 2, \ldots, l\}$. Let u be a word in $(\{0,1\}^m)^+$, with $u = u_1 u_2 \cdots u_l$ and define the morphism $\bar{u} : \mathbb{N}_l^* \to \{0,1\}^m, \bar{u}(i) = u_i$ for all $i \in \mathbb{N}_l$. Let $u_k = \omega$ for all $k \in \mathbb{N}_l \setminus \mathrm{alph}(\alpha)$ for some fixed $\omega \in \{0,1\}^m$. Now, by definition we have

$$\bar{u}(\alpha) = u_{a_1} u_{a_2} \cdots u_{a_s} .$$

Thus $\bar{u}(\alpha)$ is the word where the blocks $u_1, u_2, \ldots, u_l$ of u are written in the order and multiplicity determined by α. If k is an element of $\mathbb{N}_l = \{1, 2, \ldots, l\}$ but does not belong to $\mathrm{alph}(\alpha)$, then the respective u_k is equal to some constant message block ω. This also means that the message block u_k does not affect the computation of the hash value, i.e one can have any value for u_k without changing the resulting hash value.

We now define the *iterated compression function* $f_\alpha : \{0,1\}^n \times (\{0,1\}^m)^+ \to \{0,1\}^n$ (based on α and f) by $f_\alpha(y,u) = f^*(y, \bar{u}(\alpha))$ for each $y \in \{0,1\}^n$ and $u \in (\{0,1\}^m)^+$. It is clear from the definition that if $\alpha = \alpha_1 \alpha_2$, where $\alpha_1, \alpha_2 \in \mathbb{N}_l^*$, then $f_\alpha(y,u) = f_{\alpha_2}(f_{\alpha_1}(y,u),u)$ for each $y \in \{0,1\}^n$ and $u \in (\{0,1\}^m)^+$.

Given $k \in \mathbb{N}_+$ and $y_0 \in \{0,1\}^m$, a *k-collision (with initial value y_0) in the iterated compression function* f_α is a set $A \subseteq \{0,1\}^{ml}$ such that $|A| = k$ and for all $u, v \in A$, $|u| = |v|$ and $f_\alpha(y_0, u) = f_\alpha(y_0, v)$. Note that arbitrarily large trivial multicollisions in f_α exist. Namely, for each $r \in \mathbb{N}_+$, the set of messages of the form $(\omega)^t \{0,1\}^{mr}$ is a

2^r-collision in f_α (with any initial value), where $t = \max\{i \mid i \in \mathrm{alph}(\alpha)\}$. We say that A is *nontrivial* if for each $u = u_1 u_2 \cdots u_l$ in A such that $u_i \in \{0,1\}^m$ for $i = 1, 2, \ldots, l$, the equality $u_j = \omega$ holds for each $j \in \mathbb{N}_l \setminus \mathrm{alph}(\alpha)$. This means that the only message blocks that can be varied in the multicollision are the ones that actually affect the resulting value of the compression function.

Finally, we are ready to characterise a generalised iterated hash function. For each $l \in \mathbb{N}_+$, let $\alpha_l \in \mathbb{N}_l^+$ be such that $\mathrm{alph}(\alpha_l) = \mathbb{N}_l$. Denote $\hat{\alpha} = (\alpha_1, \alpha_2, \ldots)$. Define the *generalised iterated hash function* $H_{\hat{\alpha},f} : \{0,1\}^n \times (\{0,1\}^m)^+ \to \{0,1\}^n$ (based on $\hat{\alpha}$ and f) as follows: Given the initial value $y_0 \in \{0,1\}^m$ and the message x, which has a block representation consisting of l blocks, let $H_{\hat{\alpha},f}(y_0, x) = f_{\alpha_l}(y_0, x)$.

It is worthwhile to notice that the traditional iterated hash function $H : (\{0,1\}^m)^+ \to \{0,1\}^n$ based on f (with initial value $y_0 \in \{0,1\}^n$) can of course be defined by $H(u) = f^*(y_0, u)$ for each $u \in (\{0,1\}^m)^+$. On the other hand, H is a generalised iterated hash function $H_{\hat{\alpha},f} : \{0,1\}^n \times (\{0,1\}^m)^+ \to \{0,1\}^n$ based on $\hat{\alpha}$ and f, where $\hat{\alpha} = (1, 1 \cdot 2, 1 \cdot 2 \cdot 3, \ldots)$ and the initial value is fixed to y_0.

Given $k \in \mathbb{N}_+$ and $y_0 \in \{0,1\}^m$, a *k-collision* in the generalised iterated hash function $H_{\hat{\alpha},f}$ is a set $A \subseteq (\{0,1\}^m)^+$ such that $|A| = k$ and for all $u, v \in A$, $|u| = |v|$ and $H_{\hat{\alpha},f}(y_0, u) = H_{\hat{\alpha},f}(y_0, v)$. Suppose now that A is a k-collision set in $H_{\hat{\alpha},f}$ with the initial value y_0. Let $l \in \mathbb{N}_+$ be such that $A \subseteq (\{0,1\}^m)^l$, i.e. the messages in A consist of l message blocks. Then, by definition, for each $u, v \in A$, the equality $f_{\alpha_l}(y_0, u) = f_{\alpha_l}(y_0, v)$ holds. Since $\mathrm{alph}(\alpha_l) = \mathbb{N}_l$, the set A is a nontrivial k-collision in f_{α_l} with the initial value y_0.

A *k-collision attack on a* $H_{\hat{\alpha},f}$ is a probabilistic process (often based on the birthday problem) that finds a k-collision in $H_{\hat{\alpha},f}$ for any initial value y_0. The *complexity of a k-collision attack algorithm on* $H_{\hat{\alpha},f}$ is the expected number of queries on f required to get a k-collision.

In III and V, we explore the ideas presented by Hoch & Shamir (2006) and Nandi & Stinson (2007). In V, we provide some new definitions and a more accurate analysis of the generalised iterated hash functions. In this thesis, we present the main schema and discuss the complexity and other properties of this design. Detailed proofs are omitted and the reader is instructed to read V for a thorough exposition of this idea.

The main idea behind the multicollision attack against generalised iterated hash functions is the *Nested Multicollision Attack Schema* (NMCAS), which is the following:

The Nested Multicollision Attack Schema

Input: A generalised iterated hash function $H_{\hat{\alpha},f}$, an initial value $y_0 \in \{0,1\}^n$ and $r \in \mathbb{N}_+$.

Output: A 2^r-collision in $H_{\hat{\alpha},f}$.

1. Choose a suitably large $\ell \in \mathbb{N}_+$. Let $\alpha_\ell = i_1 i_2 \cdots i_s$ be the ℓ^{th} element of the sequence $\hat{\alpha}$, with $s \in \mathbb{N}_+$ and $i_j \in \mathbb{N}_\ell$ for all $j = 1,2,\ldots,s$.
2. Find and fix a suitably large set of *active indices* $A \in \mathbb{N}_\ell$.
3. Find a suitable factorisation of the word α_ℓ, i.e. $p \in \{1,2,\ldots,s\}$ and $\beta_i \in \mathbb{N}_\ell^+$ such that $\alpha_\ell = \beta_1\beta_2\cdots\beta_p$.
4. Using the set of active indices, generate a multicollision in f_{β_1}, i.e. message block sets $M_1, M_2, \ldots, M_\ell$, for which the following conditions hold:
   - For all $i \in \mathbb{N}_\ell \setminus A$, the set M_j consists of only one constant message block ω.
   - For all $j \in A$, the set M_j consists of two distinct message blocks $x_{j,1}$ and $x_{j,2}$.
   - The set $M = M_1 M_2 \cdots M_\ell$ is a $2^{|A|}$-collision in f_{β_1} with the initial value y_0.
5. Based on the set $C_1 = M$, find message sets $C_2, C_3, \ldots, C_p$ for which the following conditions hold:
   - $C_p \subseteq C_{p-1} \subseteq \cdots \subseteq C_1$.
   - For all $t \in \{1,2,\ldots,p\}$, the set C_t is a multicollision in $f_{\beta_1\beta_2\cdots\beta_t}$ with the initial value y_0.
   - $|C_p| = 2^r$
6. Output C_p.

If the NMCAS is carried out successfully, the set C_p is a 2^r-collision in $H_{\hat{\alpha}.f}$. Of course, proving that each of the steps in the NMCAS can be carried out and that these steps are not too complex is not an easy task. In their paper, Hoch & Shamir (2006) show that these steps can be carried out, but they do not formalise this method in this fashion. In V, we have been able to give exact formulations of the results required to prove the correctness and feasibility of the NMCAS and give a detailed analysis of its complexity.

The main idea behind the NMCAS is finding a suitable set of message blocks, which form a chain in the partially ordered set $(\mathrm{alph}(\alpha), \prec_\alpha)$. If such a chain is found, the factorisation step yields only one factor and the set of active indices consists of the corresponding indices of the elements in the chain. This idea has been proposed already

by Nandi & Stinson (2007). If such a chain cannot be found, the situation becomes more complex and the results of Hoch & Shamir (2006), III and V are needed to ascertain that even in these cases we can carry out the NMCAS.

The main differences between the results of V and the results of Hoch & Shamir (2006) are in the resulting complexity. We denote by $q \in \mathbb{N}_+$ the maximum number of occurrences of any given message block in the computation of the hash value. The results of Hoch & Shamir (2006) imply a complexity of $r_q(k^3 n^{3(q-3)+2})^{s_q} 2^{\frac{n}{2}}$ for a 2^k-collision, with r_q and s_q defined inductively as $r_1 = s_1 = 1$, $r_{i+1} = (i-1)^{s_i+1} r_i^{s_i+1}$ and $s_{i+1} = s_i^2 + 1$. In V, Theorem 5.4 states that the complexity of finding a 2^k-collision with the NMCAS is $2.5 \cdot r_q n^{(q-1)^2 s_q} k^{(2q-3)s_q} 2^{\frac{n}{2}}$, where r_q and s_q are defined inductively as $r_1 = s_1 = 1$, $r_{i+1} = i^{s_i} r_i^{s_i+1}$ and $s_{i+1} = s_i^2 + 1$.

The difference between these results can be explained by the more rigorous treatment of the words α_l in the sequence $\hat{\alpha}$. Our method leads to greater complexity, but it is formally correct and works in all possible cases. The method of Hoch & Shamir (2006) leaves some of the borderline cases unexamined. It is also worth noting, that the parameter q, which limits the occurrences of the symbols in the words α_l is not included as an input to the NMCAS. As shown in the Remark 4.12 of V, the NMCAS procedure would be at least triply exponential with respect to q. This means that with even moderate values for q the NMCAS becomes too complex. However, Kortelainen *et al.* (2011) show that the complexity of this method could be reduced significantly.

The question whether all generalised iterated hash functions are susceptible to multicollision attacks is still open. It is worth noting that the sequence $\hat{\alpha}$ needs to be effectively encoded, i.e. there needs to be an efficient method for choosing and representing the words α_l of $\hat{\alpha}$. This means that completely random words cannot be used as the storage of such words becomes infeasible quite fast. Thus, the efficient encoding has to give a rule for generating the words in the sequence and this rule might also lead into more efficient multicollision attacks. In any case, the hash function should be able to process arbitrarily long messages and some limits on the words α_l seem to be inevitable if efficient algorithms are required.

One of the major weaknesses in generalised iterated hash functions is that one needs to have all of the message before its hash value may be computed. Traditional iterated hash functions can begin the computation as soon as the first message block has been received. This limits the use of generalised iterated hash functions on streaming data. If one chooses to limit the words α_l in such a way that the first message blocks are utilised heavily in the beginning and the last message blocks in the end, the NMCAS can be

easily applied as the third step (the factorisation) can be done more easily when certain regularities start to appear in the words α_l (see Nandi & Stinson (2007), III and V).

Constructing an optimised algorithm that realises the NMCAS and testing it against real hash functions would be a natural continuation of this research. However, there is some lack of motivation for this as generalised iterated hash functions have not been constructed and remain theoretical entities at the moment. Some constructions such as the Zipper hash (Liskov 2007) share ideas from these generalised iterated hash functions, but still fall short from utilising the full potential of the construction.

It is interesting to note that the results of Hoch & Shamir (2006), III and V are of a dual nature to some very famous results in combinatorics. In many combinatorial problems of words, arbitrarily long words over a fixed (finite) alphabet are considered, whereas our results concern arbitrarily long words over alphabets of increasing size with the number of occurrences of each symbol restricted. In both cases, unavoidable regularities appear as the famous results of classical combinatorics like Ramsey's Theorem and Shirshov's Theorem show for the former case. For details, see for instance the book by DeLuca & Varrichio (1999). It could be an interesting research topic to see whether these results could be used in the context of generalised iterated hash functions.

5.4 Multicollisions on graph-based hash functions

The question of multicollision attacks becomes even harder when we turn our attention to hash functions in the class $\mathscr{D}$. These functions would generalise the construction of generalised iterated hash functions even further. The results of Hoch & Shamir (2006) and Nandi & Stinson (2004) on tree-based hash functions solve the problem for a subclass of $\mathscr{D}$. In VII, we generalise this notion of tree-based hash functions and prove a slight generalisation of the results of Hoch & Shamir (2006). We also present a slight generalisation of the hash functions in $\mathscr{D}$, i.e. the graph-based hash functions.

The tree-based hash functions have been further generalised by Bertoni *et al.* (2009). In their work, the tree-based hash functions can use a different compression function at different vertices of the tree. These types of hash functions are secure up to the birthday bound when the compression functions and the trees satisfy certain conditions.

Two of the SHA-3 candidates utilise the ideas of the tree-based construction. These candidates are MD6 (Rivest 2008) and ESSENCE (Martin 2008). However, these hash functions were not among the finalists of the SHA-3 competition. In the following, we

give some new definitions that are necessary for the further study of graph-based hash functions.

The basic definitions for graphs can be found in Section 3.3 on page 30 of this thesis. Let f be a compression function of length N and block length M with $N, M \in \mathbb{N}_+$. We define $x = x_1 x_2 \cdots x_l$ to be a binary message with l blocks of length $m \in \mathbb{N}_+$. Let $\Omega = \{\omega_1, \omega_2, \ldots, \omega_d\}$ be a set of $d \in \mathbb{N}$ distinct initial values (i.e. binary strings of length $n \in \mathbb{N}_+$) and $B_l = \{x_1, x_2, \ldots, x_l\}$ be the set of message blocks of length m corresponding to the message x. We follow the terminology of Nandi & Stinson (2004) and Hoch & Shamir (2006) and define ρ to be the initial assignment function of the digraph G. We adopt the definition from Nandi & Stinson (2004): $\rho : L(G) \to B_l \cup \Omega$ and require that the image of ρ contains B_l as a subset. We define $\mathscr{G}$ to be an indexed family of pairs (G_l, ρ_l) with the property that for all $l \in \mathbb{N}$ we have that ρ_l is an initial assignment function from $L(G_l)$ to $B_l \cup \Omega$ and that $G_l = (V_l, E_l)$ is an acyclic digraph for all $l \in \mathbb{N}$.

We assign a unique index between 1 and $|V_l|$ to each vertex. For all $v \in L(G_l)$, the intermediate value is the value $\rho_l(v)$. For each vertex $v \in V_l \setminus L(G_l)$, we define the word $z = z_{i_1} z_{i_2} \cdots z_{i_k}$, where z_{i_j} is the intermediate value assigned to the vertex u_{i_j} with the property $u_{i_j} \to v$ for all $1 \leq j \leq k$ and $i_s < i_t$ for all $1 \leq s < t \leq k$. Let $v \in V_l \setminus L(G_l)$. We define the intermediate value of the graph-based hash function h at vertex v as $h(v) = f^*(z)$. Notice that the above method defines a unique way to parse the input to the compression function from the different message blocks and initial values and that $|z| = M + N$ in the binary alphabet. The hash value $h(x)$ is defined as the value assigned to the unique sink of G_l.

Furthermore, we define $\Gamma(X)$ as the multiset of all the images of the elements of $X \subseteq L(G_l)$ under ρ_l. Let $x \in B_l$. Now, $\text{freq}(x, G_l)$ is the multiplicity of x in $\Gamma(L(G_l))$. We set $\text{freq}(G_l) = \max\{\text{freq}(x, G_l) : x \in B_l\}$. Finally, we define $S(v) = |\{x \in B_l : \text{freq}(x, G[v]) \geq 1\}|$. If v is the unique sink (i.e. a root) of a graph G, then we denote $S(v) = S(G)$.

Remark. Notice that for graph-based hash functions the set of initial values can be empty. As the computation of the hash is defined by the graph structure, there is no need to specify any initial values. We will return to this property later in this thesis.

We also adopt the definition of independent message blocks from Nandi & Stinson (2004). For a digraph to be useful in the computation of a hash function, we require that it has a root even if it is not a tree.

Definition 6. Let $l \in \mathbb{N}$ and $(G_l, \rho_l) \in \mathscr{G}$. A sequence $(x_1, x_2, \ldots, x_k), k \in \mathbb{N}_+$ of message blocks is independent if there exist vertices $v_{j_1}, v_{j_2}, \ldots, v_{j_k}$ of G_l such that

1. All occurrences of x_i are in $\rho_l(L[v_{j_i}])$ for all $1 \leq i \leq k$
2. $x_i \notin \rho_l(L[v_{j_t}])$ for all $i > t$
3. v_{j_k} is the root of G_l.

In the above definition, the value k is the length of the independent sequence. We denote by $I(G)$ the maximum value for k such that there exists an independent sequence of length k in the digraph G. Notice that the order of the message blocks is relevant in the above definition. Furthermore, we observe that if G is a tree and $(x_1, x_2, \ldots, x_k)$ is an independent sequence in G (with vertices v_{i_j} numbered accordingly), then $(x_2, x_3, \ldots, x_k)$ is an independent sequence in $G - G[v_{i_1}]$, i.e. the subtree of G not containing $G[v_{i_1}]$ (see Lemma 6 of Nandi & Stinson (2004)).

If G is a tree and there exist vertices $v_1, v_2, \ldots, v_d$ such that $S(G[v_i]) = S(G), i \in \{1, 2, \ldots, d\}$ and $I(G[v_i] - (\bigcup_{j<i} G[v_j])) = S(G)$, the tree G is said to have d independent subtrees. This definition corresponds to the "successive permutations" case in Hoch & Shamir (2006).

Let $l \in \mathbb{N}, (G_l, \rho_l) \in \mathscr{G}$ and $X \subseteq B_l$ with $G_l = (V, E)$. We define $G|_X = (V', E')$ as the subgraph of G_l such that $V' = \{v \in V : u \in L(G_l), x \in X, \rho_l(u) = x, u \Rightarrow v\}$ and $E' = \{(u, v) \in E : u, v \in V'\}$. We call $G|_X$ the *X-reachable* subgraph of G_l, i.e. the subgraph of G_l that contains all the vertices which are reachable from the leafs that have labels from the set X. The definition of $G|_X$ is equivalent with the Definition 13 of Hoch & Shamir (2006). Notice that if G_l is a tree, then the X-reachable subgraph of G_l is also a tree.

5.4.1 Tree-based hash functions

Nandi & Stinson (2004) and Hoch & Shamir (2006) prove that for binary tree-based hash functions, there exists a similar multicollision finding algorithm as for the generalised iterated hash functions. In VII, we show a fairly straightforward method for extending these methods to t-ary tree-based hash functions. In the following, we state the results from VII without proofs, which can be found in the original paper and are also easily generalised from the corresponding results of Nandi & Stinson (2004) and Hoch & Shamir (2006).

The first result is a generalisation of Lemma 6 from Nandi & Stinson (2004). The proof is identical to the original when we replace the constant 2 with the variable t. The second result generalises Lemma 6 from Hoch & Shamir (2006) to the t-ary tree case. Finally, the third result uses the two previous ones to prove a generalisation of Lemma 7 from Hoch & Shamir (2006).

Lemma 1. *Let $l \in \mathbb{N}, (G_l, \rho_l) \in \mathscr{G}$, $G_l = (V,E)$ a directed t-ary tree and r be the root of G_l. Let C be an arbitrary positive integer satisfying $\frac{S(r)}{t} \geq C$. Then, there exists a vertex $v \in V$ such that $C \leq S(v) \leq tC$.*

Lemma 2. *Let $l \in \mathbb{N}, (G_l, \rho_l) \in \mathscr{G}$ such that $G_l = (V,E)$ is a directed t-ary tree with a root r and $freq(G_l) \leq q$. Let C and D be arbitrary positive integers satisfying $\frac{S(r)}{t} \geq CD$. Then, at least one of the following conditions holds:*

1. *$I(G_l) \geq D$ or;*
2. *there exists a vertex v and $X \subseteq \rho(L(G_l))$ such that $freq(G_l[v]|_X) \leq q-1$ and $S(G_l[v]|_X) \geq C$.*

Lemma 3. *Let $l \in \mathbb{N}, (G_l, \rho_l) \in \mathscr{G}$ such that $G_l = (V,E)$ is a directed t-ary tree with $S(G_l) \geq (tk-1)x$ and $freq(G_l) = 1$. Then, there exist k distinct vertices $v_{i_1}, \ldots, v_{i_k}$ such that $L(G_l[v_{i_s}]) \not\subseteq L(G_l[v_{i_t}])$ when $s > t$ and $S(G_l[v_{i_s}] - \bigcup_{i_t < i_s} G_l[v_{i_t}]) \geq x$.*

The above results can be used to reduce the tree-based case back to the sequential case. First of all, Lemma 1 shows that we have a vertex in the tree that is limited by suitable values from above and below. Then, Lemma 2 shows that we again have two distinct cases: one with a large enough independent set of message blocks and another where the tree can be made "smaller" and thus an induction argument can be applied. Finally, Lemma 3 shows that each of the independent subtrees that the tree can be divided into has enough message blocks in the leaves' labels to enable a birthday attack on each of the subtrees when the tree is large enough and has a suitably large number of distinct message blocks as labels on the leaves.

Thus, the tree structure induces a sequential structure on the message blocks that the leaves are labelled with. Now, we can apply the results of previous section with generalised iterated hash functions and obtain a fairly similar limit for the multicollision attack on a t-ary tree-based hash function.

5.4.2 Graph-based hash functions and multicollisions

The tree-based hash functions presented in the previous section are a generalisation of the generalised iterated hash functions. Nandi & Stinson (2004) propose a class of hash functions $\mathscr{D}$ that contains the generalised iterated and tree-based hash functions as special cases. Recall that if $x = x_1x_2\cdots x_l$ is a binary message with l blocks, we have $B_l = \{x_1, x_2, \ldots, x_l\}$ and $\Omega = \{\omega_1, \omega_2, \ldots, \omega_d\}$ is a set of $d \in \mathbb{N}$ initial values. Let $s \in \mathbb{N}_+$. We define the set of precursors of an intermediate hash value y_i as $P_i = \{y_1, y_2, \ldots, y_{i-1}\}$ and the set of all intermediate values $P = \{y_1, y_2, \ldots, y_s\}$. The hash functions in the class $\mathscr{D}$ have the following properties: The computation of such a hash function is characterised by a list of triples $L = \{(t_i, x_i, y_i) : i \leq s\}$, which satisfy for all $i \in \{1, 2, \ldots, s\}$:

$$\begin{aligned} y_i &= f(t_i, x_i) \\ t_i &= t_{(i,1)}t_{(i,2)}\cdots t_{(i,c)} \\ t_{(i,j)} &\in \Omega \cup P_i \\ x_i &= x_{(i,1)}x_{(i,2)}\cdots x_{(i,b)} \\ x_{(i,j)} &\in B_l, \end{aligned}$$

where $f : \{0,1\}^N \to \{0,1\}^n$ is a compression function. If the length of the message blocks is m and the length of the initial values n, we have that $cn + bm = N$ for all i. In the above, all y_i are called intermediate hash values and the value y_s is the hash value of x (Nandi & Stinson 2004).

We define a slightly more general class of hash functions $\mathscr{D}^+$ in the following.

Definition 7. Let n, N and $s \in \mathbb{N}_+$, $n < N$ and x be a binary message of l blocks. Let $f : \{0,1\}^N \to \{0,1\}^n$ be a compression function. A hash function $h_{s,f}$ from the family $\mathscr{D}^+$ is defined as follows: Let $u_i \in \{0,1\}^N$, $u_i = v_1v_2\cdots v_{d_i}$ with $v_j \in \Omega \cup B_l \cup P_i$ for all $1 \leq j \leq d_i$ and $i \in \{1, 2, \ldots, s\}$. Now, $y_i = f(u_i)$ for all $1 \leq i \leq s$ and $h_{s,f}(x) = y_s$.

It should be noticed from the above definition that even if the words u_i are binary strings of length N, we consider these words over the alphabet $\Omega \cup B_l \cup P_i$. Thus, we are able to distinguish between words that might have the same binary representation but which are formed from different members of the different sets, i.e. we might have $\omega_i = y_j$ for some i, j, but in the computation of the intermediate hash values these values

have different implications to the multicollision attacks. It is also worth mentioning that even if no restriction on the number of steps s is given in the definition, it should be efficient to compute the hash value from any given message and thus the number of steps should be polynomial in the length of the message. Definition 7 generalises class $\mathscr{D}$ by Nandi & Stinson (2004) by allowing free interleaving of message blocks, initial values and intermediate values in the computation of the hash value.

The representatives of the class $\mathscr{D}^+$ can also be described as graphs. These graphs turn out to be graphs that have a somewhat more general structure than the t-ary trees described previously. However, some of the results also apply to these graphs.

There are two ways to approach these hash functions. First, we describe the members of the class $\mathscr{D}^+$ as the generalisations of the tree-based hash functions. This method relies on the idea that the graph which the computation is based on is given and the labelling function sets the message blocks and initial values on the sources of the graph. The other way to approach these hash functions is to look at the definition of $\mathscr{D}^+$ and to build the graphs from there.

Let $(G_l, \rho_l) \in \mathscr{G}$ with $l \in \mathbb{N}$. Let $q \in \mathbb{N}_+$ be the maximum number of occurrences of a single message block in the computation of the hash value. The *computation graph* G_l of $h_{s,f} \in \mathscr{D}^+$ is an acyclic digraph, with the following properties: The maximum outdegree of sources that have labels from B is q and the minimum is 1. There is a unique sink r and $h_{s,f}(x) = h(r)$. For all vertices of G_l which are not sources, the indegree is $p = c + b$, where $c, b \in \mathbb{N}, cn + bm = N$ and $p \geq 2$ and there are s of these vertices. Each of these vertices corresponds to a step in the computation of the hash value of the message x and these are ordered by the number $i \in \{1, 2, \ldots, s\}$ with r having the number s. There may not be an edge from an intermediate vertex to another intermediate vertex of a higher order.

Let γ be the maximum indegree of the vertices in G_l. Then, the maximum number of edges in the graph G_l is γs. The maximum number of vertices in the graph G_l is $s + ql + wd$ where w is the maximum number of occurrences of a given initial value in the computation of the hash value.

As the previous analysis of the tree-based hash functions is based on the computation graphs, it would be essential to generalise the results on tree-based hash functions to the more general computation graphs of hash functions. It is possible to generalise Lemma 1 to the computation graphs of graph-based hash functions.

Lemma 4. *Let $(G_l, \rho_l) \in \mathscr{G}$ with $l \in \mathbb{N}_+$ and r be the root vertex of G_l. Furthermore, let G_l be a computation graph of a hash function $h_{s,f} \in \mathscr{D}^+$. Let γ be the maximum indegree of the vertices in G_l. Let C be an arbitrary positive integer satisfying $\frac{S(r)}{\gamma} \geq C$. Then, there exists a vertex $v \in G_l$ such that $C \leq S(v) \leq \gamma C$.*

The proof of Lemma 4 is exactly the same as the proof of Lemma 1 with t replaced by γ. The arguments about the indegrees of the vertices are still valid although the graph is not exactly a tree and thus the claim holds.

Unfortunately, Lemmas 2 and 3 do not directly generalise to the computation graphs of hash functions from $\mathscr{D}^+$. However, because the indegree of all intermediate vertices is bound by γ and the total number of edges is bound by γs, it seems possible to find an effective multicollision attack against these types of hash functions. This is one interesting topic for further research.

Another way to study these hash functions and their properties is to build a graph from the different dependencies that the computation induces between different message blocks, initial values and intermediate values.

Definition 8. Let $h_{s,f}$ be a hash function from the family $\mathscr{D}^+$ as in Definition 7. The dependency graph of $h_{s,f}$ is a graph $G = (V, E)$ with $V = \Omega \cup B_l \cup P$ and $(a, b) \in E$ iff $a, b \in V$, $b = f(u_i)$ and $a \in \{v_1, v_2, \ldots, v_{d_i}\}$.

The dependency graph shows how the different message blocks, initial values and intermediate values interact when the hash value of a message of a given block length is computed. There are some restrictions on the graph that arise from the definition of the hash function. The constant q is an upper bound for the outdegree of all the vertices from B_l. Let β be the maximum indegree of all the vertices of the dependency graph G. The maximum amount of edges in G is then βs. The maximum amount of vertices in the dependency graph is $d + l + s$. Further study of the properties of dependency graphs is a topic for future research.

The results concerning graph-based hash functions show that even this generalisation does not provide much security against multicollisions. The graph-based hash functions suffer also from a similar drawback as the generalised iterated hash functions, as for each message length, there needs to be a graph representing the computation of the hash value for the messages of this length. Thus, the representations of these graphs have to be effectively encoded and this might lead to more predictable behaviour of the hash function, which in its turn would further weaken the construction. In the design of MD6 by Rivest (2008), the requirement of effective encoding led to the decision of having an

option where the graph structure is replaced by a sequential structure. This could then be used in some resource constrained devices. Of course, this property could be used by attackers if this sequential structure is seen as less secure.

As stated in the remark on page 71, graph-based hash functions do not necessarily require any initial values for computation. This distinguishes them from sequential hash functions, where an initial value must be given in order to compute the hash function. In graph-based hash functions, there is also a possibility to use more than one initial value, but there is very little use in that as it increases the amount of computation without introducing uncertainty. Of course, these initial values may be necessary to complete the graph, if the graph needs some predefined structure that cannot be provided with just the message blocks and intermediate values. Also, some specific constructions may require the use of several initial values. This is the case for example with the HMAC message authentication codes, where there are two initial values utilised in the computation.

In one respect, the graph-based hash functions are more effective than sequential ones. With the sequential structure, one cannot compute the hash value until the next message block is available. In graph-based hash functions, some amount of parallelisation can be utilised and the hash values for each vertex v can be computed as soon as the values for all vertices $u \to v$ have been determined. This could make them more attractive from a practical point of view as this property might allow some form of streaming computation of the hash value, which is possible for traditional iterated hash functions. In the MD6 design, the graphical structure together with parallelisation led to a fairly efficient computation of the hash function.

One possibility to further generalise the hash functions from the class $\mathscr{D}^+$ would be to allow even more general graphs as the form of computation. For example, removing the requirement for a single root vertex, would be one generalisation. The values of the different root vertices could be catenated to form the final hash value. This would be much like in the wide-pipe construction for regular iterated hash functions. Studying the properties of these hash functions could be one possibility for future research.

The generalisations presented in Bertoni *et al.* (2009) also show a possibility of constructing secure hash functions with several different compression functions used during the computation and the class $\mathscr{D}^+$ does not contain this possibility. In a more general setting, these several compression functions could be taken into account. It is our opinion that the graphical structure and the restrictions imposed by that still enable efficient multicollision attacks. This is due to the fact that the results of Nandi & Stinson (2004), Hoch & Shamir (2006) and this thesis do no rely on any specific weakness of the

underlying compression function. Thus, using many different compression functions would not necessarily provide much added protection against these methods.

6 Summary and conclusions

In this thesis, we have considered the security of hash functions with two different research problems. The first research problem was the cryptanalysis of the VSH hash function, especially against preimage attacks. This provided some new insight into the hash function and confirmed some of the previous results in a more realistic setting. The second problem was to analyse the multicollision attacks against iterated hash functions and generalised iterated hash functions. This research yielded improved analysis of some existing methods and some new results concerning the even more general classes of hash functions.

Chapter 4 of this thesis described the VSH hash function and the proof of collision resistance for VSH. We also discussed the preimage security of VSH. By extending the method of Saarinen (2006), we were able to show that many of the variants proposed for VSH share the weakness of the original VSH proposal against preimage attacks. Furthermore, we showed that realistic password schemes secured with VSH can be attacked with these preimage finding methods. These attacks were both feasible and reliable even without any additional optimisation by using dictionary searches or parallelisation.

Chapter 5 described multicollisions and Joux's method for finding multicollisions in iterated hash functions. We also presented some other methods and an extension to the method of Joux (2004). We also considered some countermeasures against this multicollision attack. We presented generalised iterated hash functions in closer detail and introduced the Nested Multicollision Attack Schema. Furthermore, we discussed the implications and limitations of this method. In the final section, we studied the properties of tree- and graph-based hash functions. We generalised the previous multicollision results to the t-ary tree-based hash functions and discussed the properties of the more general graph-based hash functions.

It still remains an open question whether all generalised iterated hash functions have an efficient multicollision against them. We conjecture that in the most general case, i.e. without any restrictions on the multiplicity and order of the message blocks, there does not exist such a method, but for all efficiently realisable generalised iterated hash functions such a method should exist.

The results of this thesis still leave many questions open for future research. Even though the VSH and its variants are shown to be rather insecure, the development of hash functions that have an elegant mathematical form and rigorous proofs of security is still important. The results on multicollisions leave much room for improvement and some results in this direction have already been achieved. Even the NMCAS method is too complex to be realised with reasonable parameters. As mentioned earlier, the general case for generalised iterated hash functions and multicollisions is still unanswered. The results concerning graph-based hash functions are very preliminary and leave many open questions for future research.

References

Andreeva E (2010) Domain Extenders for Cryptographic Hash Functions. Ph.D. thesis, Katholieke Universiteit Leuven.

Andreeva E, Bouillaguet C, Fouque PA, Hoch JJ, Kelsey J, Shamir A & Zimmer S (2008) Second preimage attacks on dithered hash functions. Proc. 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT 2008, 270–288.

Andreeva E, Neven G, Preneel B & Shrimpton T (2007) Seven-property-preserving iterated hashing: ROX. Proc. 13th International Conference on the Theory and Application of Cryptology and Information Security – ASIACRYPT 2007, 130–146.

Andreeva E & Stam M (2011) The symbiosis between collision and preimage resistance. Proc. 13th IMA International Conference on Cryptography and Coding – IMACC 2011, 152–171.

Aumasson JP (2008) Faster multicollisions. Proc. 9th International Conference on Cryptology in India – INDOCRYPT 2008, 67–77.

Barkan E, Biham E & Shamir A (2006) Rigorous bounds on cryptanalytic time/memory tradeoffs. Proc. 26th Annual International Cryptology Conference – CRYPTO 2006, 1–21.

Bayer D, Haber S & Stornetta WS (1993) Improving the efficiency and reliability of digital time-stamping. In: Capocelli RM, Santis AD & Vaccaro U (eds) Sequences II: Methods in Communication, Security, and Computer Science, 329–334. Springer.

Bellare M & Namprempre C (2000) Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. Proc. 6th International Conference on the Theory and Application of Cryptology and Information Security – ASIACRYPT 2000, 531–545.

Bellare M & Ristov T (2008) Hash functions from sigma protocols and improvements to VSH. Proc. 14th International Conference on the Theory and Application of Cryptology and Information Security – ASIACRYPT 2008, 125–142.

Bellare M & Rogaway P (1993) Random oracles are practical: a paradigm for designing efficient protocols. Proc. 1st ACM conference on Computer and communications security – CCS 1993, 62–73.

Bertoni G, Daemen J, Peeters M & Van Assche G (2009) Sufficient conditions for sound tree and sequential hashing modes. Cryptology ePrint Archive, Report 2009/210. URL: `http://eprint.iacr.org/`. Cited 2012/08/24.

Biham E, Chen R, Joux A, Carribault P, Lemuet C & Jalby W (2005) Collisions of SHA-0 and reduced SHA-1. Proc. 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT 2005, 36–57.

Biham E & Dunkelman O (2007) A framework for iterative hash functions - HAIFA. Cryptology ePrint Archive, Report 2007/278. URL: `http://eprint.iacr.org`. Cited 2012/08/24.

Black J, Rogaway P & Shrimpton T (2002) Black-box analysis of the block-cipher-based hash-function constructions from PGV. Proc. 22nd Annual International Cryptology Conference – CRYPTO 2002, 320–335.

Blake IF & Shparlinski IE (2007) Statistical distribution and collisions of the VSH. Journal of Mathematical Cryptology 1(4): 329–349.

Canetti R, Goldreich O & Halevi S (1998) The random oracle methodology, revisited. Proc. 30th Annual ACM Symposium on Theory of Computing – STOC 1998, 209–218.

Contini S, Lenstra AK & Steinfeld R (2006) VSH, an efficient and provable collision-resistant hash function. Proc. 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT 2006, 165–182.

Coppersmith D (1986) Another birthday attack. Proc. 6th Annual International Cryptology Conference – CRYPTO 1986, 14–17.

Coppersmith D, Pilpel S, Meyer CH, Matyas SM, Hyden MM, Oseas J, Brachtl B & Schilling M (1990) Data authentication using modification dectection codes based on a public one way encryption function. U.S. Patent No. 4,908,861.

Coron JS, Dodis Y, Malinaud C & Puniya P (2005) Merkle-Damgård revisited: How to construct a hash function. Proc. 25th Annual International Cryptology Conference – CRYPTO 2005, 430–448.

Crandall R & Pomerance C (2001) Prime Numbers: A computational perspective. Springer.

Damgård IB (1989) A design principle for hash functions. Proc. 9th Annual International Cryptology Conference – CRYPTO 1989, 416–427.

Davies DW & Price WL (1980) The application of digital signatures based on public-key cryptosystems. Proc. Fifth International Computer Communications Conference, 525–530.

Dean RD (1999) Formal aspects of mobile code security. Ph.D. thesis, Princeton University.

DeLuca A & Varrichio S (1999) Finiteness and Regularity in Semigroups and Formal Languages. Springer.

Diestel R (2006) Graph Theory. Graduate Texts in Mathematics. Springer.

Diffie W & Hellman ME (1976) New directions in cryptography. IEEE Transactions on Information Theory 22(5): 644–654.

Dilworth R (1950) A decomposition theorem for partially ordered sets. The Annals of Mathematics 51: 161–166.

Dobbertin H (1998) Cryptanalysis of MD4. Journal of Cryptology 11(4): 253–271.

FIPS 186-3 (2009) Digital signature standard (DSS). FIPS 186-3, National Institute of Standards and Technology.

Flajolet P & Odlyzko A (1990) Random mapping statistics. Proc. 8th Annual International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT 1989, 329–354.

Gauravaram P & Kelsey J (2007) Cryptanalysis of a class of cryptographic hash functions. Cryptology ePrint Archive, Report 2007/277. URL: `http://eprint.iacr.org/`. Cited 2012/08/24.

Gauravaram P & Kelsey J (2008) Linear-XOR and additive checksums don't protect Damgård-Merkle hashes from generic attacks. Proc. The Cryptographers' Track at the RSA Conference 2008 – CT-RSA 2008, 36–51.

Gauravaram P, Millan W, Dawson E & Viswanathan K (2006) Constructing secure hash functions by enhancing Merkle-Damgård construction. Proc. 11th Australasian Conference on Information Security and Privacy – ACISP 2006, 407–420.

Girault M & Stern J (1994) On the length of cryptographic hash-values used in identification schemes. Proc. 14th Annual International Cryptology Conference – CRYPTO 1994, 202–215.

Haber S & Stornetta WS (1990) How to time-stamp a digital document. Proc. 10th Annual International Cryptology Conference – CRYPTO 1990, 437–455.

Hellman M (1980) A cryptanalytic time-memory trade-off. IEEE Transactions on Information Theory 26(4): 401–406.

Hoch JJ & Shamir A (2006) Breaking the ICE - finding multicollisions in iterated concatenated and expanded (ICE) hash functions. Proc. 13th International Workshop on Fast Software Encryption - FSE 2006, 179–194.

ISO/IEC 18014 (2009) Information technology – security techniques – time-stamping services. ISO/IEC 18014, International Organization for Standardization/International Electrotechnical Commission.

ISO/IEC 9796-2 (2010) Information technology – security techniques – digital signature schemes giving message recovery – part 2: Integer factorization based mechanisms. ISO/IEC 9796-2, International Organization for Standardization/International Electrotechnical Commission.

Joux A (2004) Multicollisions in iterated hash functions. application to cascaded constructions. Proc. 24th Annual International Cryptology Conference – CRYPTO 2004, 306–316.

Kayser RM (2007) NIST SHA-3 hash function competition announcement. Federal Register Notices 72(212): 62212–62220.

Kelsey J & Schneier B (2005) Second preimages on n-bit hash functions for much less than 2^n work. Proc. 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT 2005, 474–490.

Keränen V (1992) Abelian squares are avoidable on 4 letters. Proc. 19th International Colloquium on Automata, Languages and Programming – ICALP 1992, 41–52.

Klima V (2005) Finding MD5 collisions on a notebook PC using multi-message modifications. Cryptology ePrint Archive, Report 2005/102. URL: `http://eprint.iacr.org/`. Cited 2012/08/24.

Kortelainen J, Kortelainen T & Vesanen A (2011) Unavoidable regularities in long words with bounded number of symbol occurrences. Proc. 17th Annual International Computing and Combinatorics Conference – COCOON 2011, 519–530.

Lai X & Massey J (1993) Hash functions based on block ciphers. Proc. 11th Annual International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT 1992, 55–70.

Lamport L (1979) Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory.

Lenstra AK (1997) Using cyclotomic polynomials to construct efficient discrete logarithm cryptosystems over finite fields. Proc. Second Australasian Conference on Information Security and Privacy – ACISP 1997, 127–138.

Lenstra AK, Lenstra, Jr HW, Manasse MS & Pollard JM (1990) The number field sieve. Proc. 22nd Annual ACM Symposium on Theory of Computing – STOC 1990, 564–572.

Lenstra AK, Page D & Stam M (2006) Discrete logarithm variants of VSH. Proc. First International Conference on Cryptology in Vietnam – VIETCRYPT 2006, 229–242.

Lenstra AK & Verheul ER (2000) The XTR public key system. Proc. 20th Annual International Cryptology Conference – CRYPTO 2000, 1–19.

Leurent G (2008) MD4 is not one-way. Proc. 15th International Workshop on Fast Software Encryption – FSE 2008, 412–428.

Liskov M (2007) Constructing an ideal hash function from weak ideal compression functions. Proc. 14th Annual Workshop on Selected Areas in Cryptography – SAC 2007, 358–375.

Lucks S (2005) A failure-friendly design principle for hash functions. Proc. 11th International Conference on the Theory and Application of Cryptology and Information Security – ASIACRYPT 2005,, 474–494.

Lysyanskaya A (2002) Signature schemes and applications to cryptographic protocol design. Ph.D. thesis, Massachusetts Institute of Technology.

Martin JW (2008) ESSENCE: A candidate hashing algorithm for the NIST competition. Submission to NIST. URL: http://www.math.jmu.edu/~martin/essence/Supporting_Documentation/essence_NIST.pdf. Cited 2012/08/24.

Maurer UM, Renner R & Holenstein C (2004) Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. Proc. First Theory of Cryptography Conference – TCC 2004, 21–39.

Menezes AJ, van Oorschot PC & Vanstone SA (1996) Handbook of Applied Cryptography. CRC Press.

Merkle RC (1979) Secrecy, authentication, and public key systems. Ph.D. thesis, Stanford University.

Merkle RC (1989a) A certified digital signature. Proc. 9th Annual International Cryptology Conference – CRYPTO 1989, 218–238.

Merkle RC (1989b) One way hash functions and DES. Proc. 9th Annual International Cryptology Conference – CRYPTO 1989, 428–446.

Mishra PK & Sarkar P (2003) Inversion of several field elements: A new parallel algorithm. Cryptology ePrint Archive, Report 2003/264. URL: http://eprint.iacr.org/. Cited 2012/09/14.

Montgomery P (1985) Modular multiplication without trial division. Mathematics of Computation 44(170): 519–521.

Nandi M & Stinson DR (2004) Multicollision attacks on generalized hash functions. Cryptology ePrint Archive, Report 2004/330. URL: http://eprint.iacr.org/. Cited 2012/08/24.

Nandi M & Stinson DR (2007) Multicollision attacks on some generalized sequential hash functions. IEEE Transactions on Information Theory 53(2): 759–767.

Pomerance C (1985) The quadratic sieve factoring algorithm. Proc. 3rd Annual International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT 1984, 169–182.

Preneel B (1993) Analysis and design of cryptographic hash functions. Ph.D. thesis, Katholieke Universiteit Leuven.

Preneel B (2008) The state of hash functions and the NIST SHA-3 competition. Proc. 4th International Conference on Information Security and Cryptology – Inscrypt 2008, 1–11.

Preneel B, Govaerts R & Vandewalle J (1993) Hash functions based on block ciphers: A synthetic approach. Proc. 13th Annual International Cryptology Conference – CRYPTO 1993, 773: 368–378.

Python Software Foundation (2007) Python programming language. URL: http://python.org/. Cited 2012/08/24.

Quisquater JJ & Delescaille JP (1990) How easy is collision search. new results and applications to DES. Proc. 9th Annual International Cryptology Conference – CRYPTO 1989, 408–415.

Rabin MO (1978) Digitalized signatures. In: DeMillo RA, Dobkin DP, Jones AK & Lipton RJ (eds) Foundations of Secure Computation, 155–168. Academic Press.

Ristenpart T, Shacham H & Shrimpton T (2011) Careful with composition: Limitations of the indifferentiability framework. Proc. 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT 2011, 487–506.

Rivest RL (1991) The MD4 message digest algorithm. Proc. 10th Annual International Cryptology Conference – CRYPTO 1990, 303–311.

Rivest RL (1992) The MD5 message-digest algorithm. Internet Request for Comment RFC 1321, Internet Engineering Task Force.

Rivest RL (2005) Abelian square-free dithering for iterated hash functions. URL: `http://csrc.nist.gov/groups/ST/hash/documents/rivest-asf-paper.pdf`. Cited 2012/08/24.

Rivest RL (2008) The MD6 hash function – a proposal to NIST for SHA-3. Submission to NIST. URL: `http://groups.csail.mit.edu/cis/md6/submitted-2008-10-27/Supporting\_Documentation/md6\_report.pdf`. Cited 2012/08/24.

Rivest RL, Shamir A & Adleman LM (1978) A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21(2): 120–126. See also U.S. Patent 4,405,829.

Rogaway P & Shrimpton T (2004) Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. Proc. 11th International Workshop on Fast Software Encryption – FSE 2004, 371–388.

Rubin K & Silverberg A (2003) Torus-based cryptography. Proc. 23rd Annual International Cryptology Conference – CRYPTO 2003, 349–365.

Saarinen MJO (2006) Security of VSH in the real world. Proc. 7th International Conference on Cryptology in India – INDOCRYPT 2006, 95–103.

Stevens M (2006) Fast collision attack on MD5. Cryptology ePrint Archive, Report 2006/104. URL: `http://eprint.iacr.org/`. Cited 2012/08/24.

Stevens M, Lenstra AK & de Weger B (2007) Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. Proc. 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT 2007, 1–22.

Suzuki K, Tonien D, Kurosawa K & Toyota K (2008) Birthday paradox for multi-collisions. IEICE Transactions 91-A(1): 39–45.

Une M (2001) The security evaluation of time stamping schemes: The present situation and studies. IMES Discussion Papers Series 2001-E-18.

van Oorschot P & Wiener M (1996) Improving implementable meet-in-the-middle attacks by orders of magnitude. Proc. 16th Annual International Cryptology Conference – CRYPTO 1996, 229–236.

Van Rompay B (2004) Analysis and design of cryptographic hash functions, mac algorithms and block ciphers. Ph.D. thesis, Katholieke Universiteit Leuven.

Wang X, Yin YL & Yu H (2005) Finding collisions in the full SHA-1. Proc. 25th Annual International Cryptology Conference – CRYPTO 2005, 17–36.

Wang X & Yu H (2005) How to break MD5 and other hash functions. Proc. 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT 2005, 19–35.

Wiener MJ (2004) The full cost of cryptanalytic attacks. Journal of Cryptology 17: 105–124.

Yu H & Wang X (2007) Multi-collision attack on the compression functions of MD4 and 3-pass HAVAL. Proc. The 10th International Conference on Information Security and Cryptology – ICISC 2007, 206–226.

Original articles

I Halunen K, Rikula P & Röning J (2008) On the Security of VSH in Password Schemes. Proceedings of the Third International Conference on Availability, Reliability and Security (ARES 2008): 828–833.

II Halunen K, Rikula P & Röning J (2009) Finding Preimages of Multiple Passwords Secured with VSH. Proceedings of the Fourth International Conference on Availability, Reliability and Security (ARES 2009): 499–503.

III Halunen K, Kortelainen J & Kortelainen T (2010), Combinatorial Multicollision Attacks and Generalized Iterated Hash Functions. Proceedings of Australasian Information Security Conference 2010 (AISC 2010): 86–93.

IV Kortelainen T, Kortelainen J & Halunen K (2010) Variants of Multicollision Attacks on Iterated Hash Functions. Proceedings of the Sixth China International Conference on Information Security and Cryptology (Inscrypt 2010): 139–154.

V Kortelainen J, Halunen K & Kortelainen T (2010) Multicollision Attacks and Generalized Iterated Hash Functions. Journal of Mathematical Cryptology 4(3): 239–270.

VI Halunen K & Röning J (2010) Preimage Attacks Against Variants of Very Smooth Hash. Proceedings of the Fifth International Workshop on Security (IWSEC 2010): 251–266.

VII Halunen K (2011) Multicollisions and Graph-based Hash Functions. Proceedings of the Third International Conference on Trusted Systems (INTRUST 2011): 156–167.

Original publications are not included in the electronic version of the dissertation.

Book orders:
Granum: Virtual book store
http://granum.uta.fi/granum/

UNIVERSITY OF OULU P.O.B. 7500 FI-90014 UNIVERSITY OF OULU FINLAND

ACTA UNIVERSITATIS OULUENSIS

ISBN 978-951-42-9965-0 (Paperback)
ISBN 978-951-42-9966-7 (PDF)
ISSN 0355-3213 (Print)
ISSN 1796-2226 (Online)

UNIVERSITY of OULU
OULUN YLIOPISTO