Des. Codes Cryptogr. (2015) 77:611–631
DOI 10.1007/s10623-015-0096-0

Open problems in hash function security

Elena Andreeva[1,2] · Bart Mennink[1,2] · Bart Preneel[1,2]

Received: 15 October 2014 / Revised: 5 May 2015 / Accepted: 7 May 2015 /
Published online: 28 May 2015

Abstract A cryptographic hash function compresses arbitrarily long messages to digests of a short and fixed length. Most of existing hash functions are designed to evaluate a compression function with a finite domain in a mode of operation, and the compression function itself is often designed from block ciphers or permutations. This modular design approach allows for a rigorous security analysis via means of both cryptanalysis and provable security. We present a survey on the state of the art in hash function security and modular design analysis. We focus on existing security models and definitions, as well as on the security aspects of designing secure compression functions (indirectly) from either block ciphers or permutations. In all of these directions, we identify open problems that, once solved, would allow for an increased confidence in the use of cryptographic hash functions.

Keywords Hash functions · Compression functions · Reductions · Permutation based · Block cipher based · Open problems

Mathematics Subject Classification 94A60

This is one of several papers published in *Designs, Codes and Cryptography* comprising the "Special Issue on Cryptography, Codes, Designs and Finite Fields: In Memory of Scott A. Vanstone".

✉ Bart Mennink
bart.mennink@esat.kuleuven.be

Elena Andreeva
elena.andreeva@esat.kuleuven.be

Bart Preneel
bart.preneel@esat.kuleuven.be

1 Department Electrical Engineering, ESAT/COSIC, KU Leuven, Leuven, Belgium

2 iMinds, Ghent, Belgium

1 Introduction

A cryptographic hash function $\mathcal{H} : \mathbb{Z}_2^* \to \mathbb{Z}_2^n$ is a function that maps bitstrings of arbitrary length to outputs of a fixed length n bits. One of the first appearances of cryptographic hash functions is in the seminal paper of Diffie and Hellman [42] who used them to construct more efficient and compact digital signatures: instead of signing a large message, they suggested to sign its hashed value. Ever since, the use of hash functions has extended to numerous applications, such as password protection, pseudorandom bits and key generation, entropy extraction, etc. Historically, an important reason for the fast hash functions spread has been their efficiency in comparison to existing block ciphers at the time. In the 90s, the MD5 design was 10 times faster than DES in software, it was believed to offer a larger security level than DES, it could deal with variable length inputs, and it posed less problems under export control laws. As a consequence, hash functions came to be used to construct MAC algorithms, stream ciphers and block ciphers.

The design of a secure cryptographic hash function involves many aspects. Firstly, a proper security model is paramount. Following Merkle's seminal 1979 PhD thesis [92], the main security properties a hash function should satisfy are collision resistance, second preimage resistance, and preimage resistance. However, the variety of hash function applications often impose further more specific security requirements.

Once the target security properties are identified, the next step is the hash function design itself. Recall that one of the main functional requirements for a hash function is the ability to accommodate arbitrarily (or at least very) long inputs. To achieve this, most hash functions in the literature use a compression function in a mode of operation. A compression function $f : \mathbb{Z}_2^{m+n} \to \mathbb{Z}_2^n$ in many cases inherits the conventional security notions from hash functions, but it differs from a hash function in that it only allows for inputs of a fixed length. Consequently, compression functions are often easier to design and analyze, and the problem of designing a hash function is shifted to the problem of designing a compression function.

Thus, the next design step lies in the development of a secure compression function f. A basic way to do so is to build f from one or more block ciphers or one or more permutations. The former, the block cipher approach, dates back to 1978 with Rabin's compression function $f(x, y) = \text{DES}_y(x)$. A generalization of this and other constructions by Preneel et al. [105] lead to a monotonic increase in the popularity of this approach. Particularly, it found adoption in MD4 [108], MD5 [109], SHA-0/1/2, Whirlpool [14], and many others. The permutation approach has gained more attention only recently with the introduction of the sponge methodology by Bertoni et al. [22], with Keccak/SHA-3 as its most prominent example. Note that this construction does not rely on a secure compression function, as will be explained later.

Because of their wide deployment and hence paramount security importance, hash functions should be supported with a strong security foundation. Such a statement supports the claim that all hash function designs should have and come with a rigorous security analysis. Often case, however, as shown by the examples of the attacks of Wang et al. [126,127] on both MD5 and SHA-1, and the generic second preimage attacks by Dean [40] and Kelsey and Schneier [63] on the Merkle-Damgård mode of operation, security vulnerabilities may remain elusive for years.

To get a better idea of the present hash function security requirements, an exposure to the open problems in the area is highly relevant. Indeed, from a theoretical perspective, it allows us to acquire a better understanding of what hash functions we can design, and from a practical point of view, it is an early recognition, identification, and possible exclusion and/or correction of existing problems.

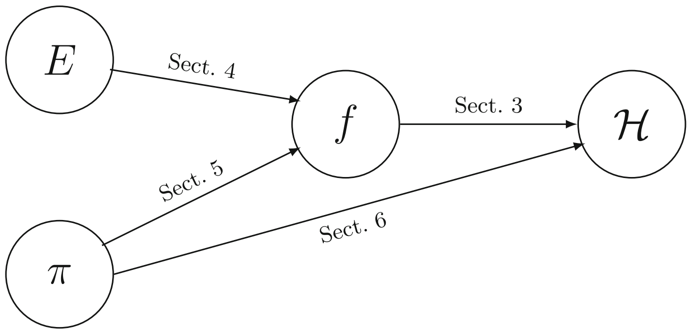


Fig. 1 Graphical description of the outline. Not depicted is Sect. 2 on hash function definitions and security models, and the conclusions from Sect. 7

In this work, we present a survey on the state of the art on hash function security and present open problems in these directions. We focus on various aspects of modular hash function design. Firstly, in Sect. 2 we consider the current scientific perspective on hash function definitions and security models, and discuss various hash function design objectives and considerations. Next, we consider the modular design process, discussing as a starting point ways to build a hash function $\mathcal{H}$ from a compression function f in Sect. 3. Also here, we discuss several open problems along the way. In Sects. 4 and 5, we discuss state of the art compression function f modular design approaches, namely building f from one or more block ciphers E or permutations π, respectively. These sections include brief discussions on all designs known to date, together with several concrete and general open problems. Finally, in Sect. 6, we summarize a direct approach on building a hash function from a permutation π. This section is motivated by the existence of various permutation-based hash functions whose underlying compression functions f are not necessarily secure, and hence do not directly fit in above approach.

The outline of this paper is graphically summarized in Fig. 1.

2 Hash function definitions and security models

For a positive integer value $n \in \mathbb{N}$, we denote by $\mathbb{Z}_2^n$ the set of bitstrings of length n, and by $(\mathbb{Z}_2^n)^*$ the set of strings of length a positive multiple of n bits. We denote by $\mathbb{Z}_2^*$ the set of bitstrings of arbitrary length. If x, y are two bitstrings, their concatenation is denoted by $x\|y$. A *hash function* is a function $\mathcal{H} : \mathbb{Z}_2^* \to \mathbb{Z}_2^n$ mapping inputs of arbitrary length to outputs of fixed bit length n. We treat (fixed input length) compression functions f and (variable input length) hash functions $\mathcal{H}$ just the same, the former being simply a special case of the latter in the way that it takes only inputs of fixed input length. The padding function $\mathsf{pad} : \mathbb{Z}_2^* \to \left(\mathbb{Z}_2^m\right)^*$ transforms the message of arbitrary length to a message with a length that is a multiple of m bits; the number of m-bit blocks is denoted by k with $k \geq 1$. A padding rule is called *suffix-free*, if for any distinct M, M', there exists no bitstring X such that $\mathsf{pad}(M') = X\|\mathsf{pad}(M)$. A padding rule is called *prefix-free*, if for any distinct M, M', there exists no bitstring X such that $\mathsf{pad}(M') = \mathsf{pad}(M)\|X$.

2.1 Standard and ideal security definitions

To be deployed in practice hash functions need to satisfy security properties which prevent attacks on the application or functionality that makes use of the functions. The main standard security properties of hash functions, collision (col), second preimage (sec) and preimage security (pre), were first studied in the 1979 PhD thesis of Merkle [92]. Informally, a hash function is collision secure if it is "hard" to find two arbitrarily chosen distinct input messages mapping to the same output hash value. When one of these input messages is fixed in advance the target security property is second preimage. Preimage security is the "hardness" of finding the preimage message for a fixed output value. These are also identified as the most relevant security notions for practical cryptographic applications and were also specified by NIST [98] as the main security requirements for the SHA-3 hash function. For a hash function with output of length n bits, one expects to find collisions with high probability after approximately $2^{n/2}$ queries (due to the birthday attack [131]). Similarly, (second) preimages can be found with high probability after approximately 2^n queries.[1] A bound is called *tight* if the lower and upper bound are the same up to a constant factor, and *optimal* if the lower bound is the trivial bound.

Recently, new *idealized* definitions have emerged and the most prominent example here is the "indifferentiability from a random oracle" notion introduced by Maurer et al. [84] and analyzed in the hash function scenario by Coron et al. [35]. A random oracle or an ideal hash function is a theoretical abstraction that models the hash function as a public function mapping inputs of arbitrary length to outputs of finite length, or $RO : \mathbb{Z}_2^* \rightarrow \mathbb{Z}_2^n$, where the outputs are drawn uniformly at random from the range space. Public here means that the function is accessible to all parties in a *black-box* manner, namely they observe only the input-output behavior of RO. Real world hash functions do not behave as random oracles because they have a well-defined structure and a compact description. The goal of the indifferentiability notion is thus to establish how far off a real world hash function with some assumed idealized structure is from an ideal hash function (random oracle). A direct consequence of the indifferentiability result [84] is that if the hash function $\mathcal{H}^P$ is based on an ideal primitive P and is indifferentiable from RO, then $\mathcal{H}^P$ can security replace RO in any 1-stage system (i.e., a system with a single 1-stage adversary). This is for example the case with signature schemes where the assumption on the hash function is that it behaves like a random oracle.

2.2 Certificational security definitions

In [85] the *certificational* hash function properties are referred to as the minimal security requirements hash functions have to satisfy. These are more intuitive design principles, which do not necessarily amount to a full-fledged attack on the hash function. Certificational security properties include: non-correlation between input and output bits; strong output propagation of input bit differences known as the avalanche property; no input bits predictability given the output bits or local one-wayness; no remaining input bits should be easy to recover given some of input bits or partial preimage resistance; pseudo-, free-start, near-collision attacks; and more.

2.3 Finding the right security definition

The main security properties of hash functions, as already indicated, are collision, second preimage, and preimage security. However, in the hash function literature one can find multiple variants of these three notions and beyond. If we take the example of preimage security, one can identify at least twelve notions: everywhere-, always-, somewhere-, domain- and

[1] Kelsey and Schneier [63] describe a second preimage attack on the Merkle-Damgård hash function that requires at most approximately 2^{n-L} queries, where the first preimage is of length at most 2^L blocks.

range-based, adaptive, chosen forced prefix, chosen target, strengthened and enhanced, partial preimage security, and preimage awareness [8,45,62,106,112,130]. It is then a valid question to ask what the right security notion for hash functions is. One solution is to pin down the concrete application and identify the concrete security properties required. This approach is however impractical because it requires a dedicated solution for the each application.

A different approach requires a close examination of the relations and separations among existing security notions, such that a set of notions with a sufficiently strong security level can be identified. Strong security notions or *super notions* which imply other (weaker) notions are desirable from practical point of view in the sense that a hash function secure with respect to the super notion also satisfies security with respect to the weaker notion. But sometimes strong security notions can be problematic when it turns out that no efficient hash function (or no hash function at all) satisfies that security property.

A recent example of such a strong notion is reset-indifferentiability [107], which was proposed as a strengthening of the indifferentiability notion and which aimed to broaden the scope of indifferentiability in various cryptographic settings. However, [81,107] show that it is impossible to build any hash function which is reset-indifferentiable.

The consequence of such a result is to aim for a weaker than reset-indifferentiability but stronger than indifferentiability notion of security. Several research papers [12,41,81,97] attempt to come up with such definitions, but not much is known yet of what the implications of these new definitions are and which designs satisfy them. One alternative, as attempted by [12], might be to establish indifferentiability results and composition theorems for only limited classes of games.

A super notion is sometimes the product of combining two or more security notions. Andreeva and Stam show in [8] that a collision and everywhere preimage secure hash function implies the notion of preimage security for sufficiently random input messages. Theoretical questions of this type remain still unaddressed and leave room for new security findings possibly along with the introduction of more efficient hash functions.

2.4 Estimating the right security complexity

A distinct but related question here is to properly identify the target complexities of the security property as a way to improve efficiency and guarantee adequate security. For example a well-known result by Rogaway and Shrimpton [112] states that finding second preimages is provably harder than finding collisions, and similar for preimages (for sufficiently large hash functions). Formally, if we denote the advantage of breaking security property $\mathsf{atk} \in \{\mathsf{col}, \mathsf{sec}, \mathsf{pre}\}$ for $\mathcal{H}$ by $\mathbf{Adv}^{\mathsf{atk}}_{\mathcal{H}}$, we have $\Omega(q/2^n) = \mathbf{Adv}^{\mathsf{sec}}_{\mathcal{H}} \leq \mathbf{Adv}^{\mathsf{col}}_{\mathcal{H}} = O(q^2/2^n)$, and $\Omega(q/2^n) = \mathbf{Adv}^{\mathsf{pre}}_{\mathcal{H}} \leq \mathbf{Adv}^{\mathsf{col}}_{\mathcal{H}} + \varepsilon$, where ε is negligible if $\mathcal{H}$ is a variable input length hash function. These results imply that if a hash function is collision secure then one obtains security guarantees also for (second) preimage security albeit up to the birthday bound. Birthday bound security, however, is suboptimal for (second) preimage security and might be insufficient for practical hash functions as identified by NIST in their SHA-3 security requirements. From a practical point of view one can wonder whether it is needed to have (second) preimage security larger than the collision security $2^{n/2}$ for reasonable values of the digest size n, since very often it is required that the security level against collisions is already way beyond reach for the next decades; a discussion around this topic arose when NIST was finalizing the specifications of the SHA-3 winner Keccak [21].

Additionally, target security levels allow to correctly interpret the tightness of the security bounds by means of reductionist security proofs.

2.5 Standard versus ideal security proofs

One of the main debates in cryptography and in hash function research in particular is on the interpretations of ideal model versus standard model security proofs. The main disadvantage of ideal proofs is that they rely on an ideal object, be it a compression function, a block cipher, or a permutation. What ideal model proofs thus show is that the hash (compression) function built from an ideal underlying primitive is sound from a structural point of view. But these proofs convey no information for the situations where attacks can be launched on the underlying primitive. In that case it is up to the cryptanalyst to verify whether the specific vulnerabilities propagate under composition. The standard model results provide a stronger guarantee in the sense that they ensure computational security via a concrete standard assumption on the primitive. Once that assumption is proven or verified for example via means of cryptanalysis, no attack on the hash (compression) function succeeds for the guaranteed standard security property. On the other hand, standard model proofs mostly focus on preservation of compression function security properties, and it often appears to be less useful for analyzing the security of hash functions based on block ciphers or permutations (we elaborate on this in Sect. 4).

2.6 Keyed hash functions

In certain scenarios as outlined below, hash functions (could) require an additional key input K. The key K in a hash function could be:

1. *Long term secret key* The key is a secret parameter shared between the parties with a legitimate access to it. Hash functions with secret key inputs are also known as message authentication codes (or MACs). We refer to the seminal works of Tsudik [125], Preneel and van Oorschot [103], and Bellare et al. [15].
2. *Long term publicly-known key* This use of the key is so far more of a theoretical than practical interest. The key in this scenario is public knowledge and its use is reminiscent of the public key scenario. However, in the hash function setting no corresponding secret key is available. This is because the public key is shared in a symmetric fashion between the communicating parties. The key in such a scenario is chosen uniformly at random and provided to all parties. The hash function $\mathcal{H}$ can be viewed as a family of hash functions where the particular instance $\mathcal{H}_K$ is fixed by the concrete choice of the key K. No hash function in use is chosen in that way.
3. *Per message/short term publicly-known key (salt)* Salting usually refers to the application of the key on a per message basis and finds use in password hashing as first suggested by Davies and Price [39] and Merkle [92]. Message salting for hashing was formalized towards constructing concrete designs such as randomized hashing to strengthen the security of signatures against collision attacks. For example, in [55] Halevi and Krawczyk proposed such designs in this direction that were later analyzed by Gauravaram and Knudsen in [51].

2.7 To key or not to key

Focusing on the latter two categories we outline some advantages and limitations of keyed hash functions:

1. *Hash function heterogeneity and improved security guarantees* The possibility of each user to fix their own hash function instance and publish the hash function key, or to fix distinct keys for specific hash function applications creates a wide range of hash functions.

In both keying per message (salting) and keying per user scenarios an adversary coming up with an attack, which works equally well for any key and which breaks a concrete hash function instance in t steps, has to perform the same amount of work to break every subsequent member of the family. This was the main motivation to introduce salting in order to increase the security of passwords against a dictionary attack. The main goal here is to avoid degradation of the security level with the usage of the hash function. Salting for the password hashing has become de facto a standard, indicative of which is the inclusion of salt as a mandatory input in the call for submissions for the recently ended competition for the design of a password hashing algorithm [102]. While salting ensures protection against dictionary attacks, it should be taken also with care if the underlying compression function suffers vulnerabilities, such as the presence of fixed points [49].

2. *Efficiency* A downside of the use of keyed hash functions is the possible efficiency loss. This becomes apparent in iterative hash function composition methods which repeatedly call the underlying keyed hash function primitives. Such designs suffer efficiency loss because in every iteration an additional key input is processed instead of additional message bits. This is an issue in settings where speed is paramount (e.g., hashing of large databases).
3. *Backwards compatibility* When introducing a keyed hash function to replace a keyless equivalent, then backwards compatibility becomes essential. A simple solution to this situation would be to fix an honestly generated and publicly-known key K (chosen, for example, by some trusted entity), and define the keyless hash function as $\mathcal{H}(M) = \mathcal{H}(K, M)$.
4. *Adversarially-chosen keys* In the long term hash key setting the security is guaranteed only if the key is honestly chosen. For common cryptographic applications of hash functions, such as digital signature schemes and key derivation mechanisms, security is indeed guaranteed for honestly generated keys. Still, in many non-standard hash function settings, the security of the hash function can be compromised in the face of adversarially-chosen keys. For example this might happen when the hash function is used as an underlying building block for another explicitly keyed cryptographic primitive. The solution for such settings would be to require a fixed, honestly-generated key by a trusted third party or to ensure that the hash function is secure even if the adversary can choose the key.

In view of these observations and mainly the specifics of the key generation process and the potential threats stemming thereof, explicitly keyed functions are not yet considered as a more "flexible" design alternative to keyless hash functions. An in-depth analysis of the implications of explicit hash function keying remains still a largely unexplored area.

3 Hash functions based on compression functions

In this section, we consider open problems related to the construction of hash functions based on compression functions. This is also known as *domain-extending* or a *mode of operation*, where a compression function f with a certain fixed and finite domain is extended to a hash function $\mathcal{H}$ with a virtually infinitely large domain.

Central to many hash function designs is the *iterated hash function principle* [70]: on input of an initialization vector IV, the iterated hash function $\mathcal{H}^f$ based on the compression function f processes a padded message $(M_1, \ldots, M_k)$ as follows:

$$\begin{aligned} \mathcal{H}^f(\mathsf{IV}; M_1, \ldots, M_k) = h_k, \text{ where:} & \\ h_0 &= \mathsf{IV}, \\ h_i &= f(h_{i-1}, M_i) \quad \text{for} \quad i = 1, \ldots, k. \end{aligned}$$

This principle is also called the plain Merkle-Damgård (MD) design [37,91]. The specific Merkle-Damgård padding rule ensures suffix-freeness by appending the message length in the last message block. The Merkle-Damgård principle was adopted by popular hash functions including SNERFU [91], MD4 [108], MD5 [109], RIPEMD [43], HAVAL [132], WHIRLPOOL [14], the SHA [116] family, and numerous other hash functions. Additionally, other domain extenders such as HAIFA [25] and dither hash [110] extend the Merkle-Damgård iterative approach by including specific counters or tweak inputs to the compression function.

3.1 Standard model security results

As hash functions and compression functions conceptually target the same goals, security analysis has been traditionally performed in the *standard model*. This means that the security of $\mathcal{H}$ is expressed in terms of the standard security of f. The most common goal here is to exhibit a *preservation* proof; proving that if f is atk secure, then $\mathcal{H}$ is atk secure where $\mathsf{atk} \in \{\mathsf{col}, \mathsf{sec}, \mathsf{pre}\}$.

An important consequence of the use of suffix-free padding together with a fixed IV value iteratively as done in the plain MD (also called MD-strengthening [70]) is the preservation of the collision security [37,91]. The collision preservation result is generalized by [3] to apply for iterative domain extenders finalized by a distinct compression function G and/or a final transformation. Other security properties, such as second preimage and preimage security, are however not preserved in the MD design [6]. Andreeva et al. [6] explore the atk property preservation of eleven domain extenders to find out that in fact none preserves all. This raises the open question whether it is possible at all to preserve the three main security properties in the standard model. More interestingly, none of the domain extenders preserves preimage security, resulting in the open problem whether preimage security alone is feasible in the standard model. The domain extender ROX [6] is an example of an all property preserving hash function with the caveat that while the assumptions on the compression function are kept in the standard model, the construction requires the use of special state and padding masks which are generated via an ideal function. From a theoretical point of view, explicitly keyed domain extenders (keyed hash functions where the key is treated as a public input) score better with respect to satisfying standard model security notions.

We refer to Andreeva et al. [2] for a survey of various domain extenders in literature, along with a thorough discussion of which security properties are preserved by these modes. They also highlight certain open problems in the direction of domain extenders. In the remainder of this work, we will focus on open problems in hash function design and modular design analysis *beyond* the work of [2].

3.2 Ideal model security results

In the ideal model one assumes that the underlying compression function(s) f of the domain extender $\mathcal{H}$ is an ideal function, namely a random oracle with a fixed input and output length. These are commonly referred to as generic security results, which ensure that the hash function has no structural defects.

There exist two types of security results. One type of results relates to proving a standard security property atk of $\mathcal{H}$ when f is an ideal function and $\mathsf{atk} \in \{\mathsf{col}, \mathsf{sec}, \mathsf{pre}\}$. Additionally, we have indifferentiability [35,84] security results where the objective is to show that $\mathcal{H}$ itself behaves like a random oracle on arbitrary input lengths given that f is an ideal function. Indifferentiability is a stronger form of indistinguishability (or pseudorandomness), which is only a meaningful notion when the algorithms are keyed (e.g. block ciphers). Indifferentiability comes therefore handy in the hash function setting most notably to allow one to obtain a bound on the adversarial advantage against some atk for $\mathcal{H}$ denoted as $\mathbf{Adv}_{\mathcal{H}}^{\mathsf{atk}}$: for any hash function security notion atk: $\mathbf{Adv}_{\mathcal{H}}^{\mathsf{atk}} \leq \mathbf{Pr}_{RO}^{\mathsf{atk}} + \mathbf{Adv}_{\mathcal{H}}^{\mathsf{pro}}$, where $\mathbf{Pr}_{RO}^{\mathsf{atk}}$ denotes the success probability of a generic attack against $\mathcal{H}$ behaving like a random oracle under atk. This bound is proven in [3].

Coron et al. [35] prove that the MD design with a suffix-free padding and idealized compression function is not indifferentiable from a random oracle. Their observation formalizes the length extension attack: one can compute $\mathcal{H}(M\|M')$ from $\mathcal{H}(M)$ and M' even without knowing M, which is highly undesirable for some applications (note that for simplicity abstraction is made here from padding, but this can be easily addressed). However, the MD construction *is* indifferentiable if it ends with a chopping function (where a portion of output bits are truncated) or a final transformation, both when the underlying compression function is ideal or when the hash function is based on an PGV compression function (see Sect. 4). Also the MD design based on ideal compression function or idealized PGV construction, with prefix-free padding has been shown to be indifferentiable from a random oracle [33,35,53,80]. Security notions such as collision resistance, are however not preserved in the MD design with prefix-free only padding in the standard model. On the other hand, domain extenders that score well on multiple property preservation, such as BCM [7] and XOR-(linear) hash [16] still lack a security analysis under the indifferentiability notion.

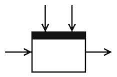
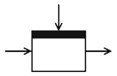
3.3 New domain extenders

Novel hash domain extenders based on compression functions need to come with at least comparable security and performance to SHA-2 for the respective digest sizes. Breaking the sequential structure is one direction and results in designs of a tree type. Known collision preserving tree-based hash functions are the (strengthened) Merkle tree [91], XOR-tree hash [16], and a variant of the latter second preimage security preserving trees [78,79,115]. The second round SHA-3 candidate hash function Skein also comes with a tree variant [47]. Finally, we mention the tree hash functions of Rivest et al. [111] and Bertoni et al. [18], where the latter also analyzed the required properties of tree hash functions to obtain indifferentiable designs. Various directions exist in this area. Some of them include analysis of tree-based alternatives or generic transformations which turn a ready-made sequential domain extenders into their parallel(-izable) counterpart. A similar approach was taken in the Sakura construction by Bertoni et al. [23].

4 Compression functions based on block ciphers

To build secure domain extenders, we not only need to show security of composition, but also need to construct secure underlying primitives as a validation of our initial assumptions. Here, we focus on the security of compression functions as underlying hash function primitives. The traditional approach is to build compression functions on one or more block ciphers. Since the late 70s, this methodology has become the dominating approach in hash function design and plenty of hash functions have been constructed accordingly. In [105], Preneel, Govaerts, and Vandewalle (PGV) analyzed and classified the 64 most natural block cipher based $2n$-to-n-bit compression functions. Black et al. [28] formally proved twelve of the PGV functions collision and preimage secure in the ideal cipher model, including the well-known Matyas-Meyer-Oseas [83], Miyaguchi-Preneel, and Davies-Meyer [94] compression

Table 1 Asymptotic ideal cipher model security guarantees of double length compression functions in the classes DBL^{2n} (first) and DBL^{n} (second)

Compression function	E-calls	Collision security	Preimage security	Indifferen-tiability	Underlying cipher
Stam [119,120]	1	2^n	2^n	1 [87]	
Tandem-DM [70]	2	2^n [76]	2^{2n} [9,77]	1 [87]	
Abreast-DM [70]	2	2^n [48,73]	2^{2n} [9,77]	1 [87]	
Hirose [58]	2	2^n	2^{2n} [9,77]	1 [87]	
Hirose's class [57]	2	2^n	$2^n \,..\, 2^{2n}$	1 [87]	
Özen-Stam's class [99]	2	2^n	$2^n \,..\, 2^{2n}$	1 [87]	
MDC-2 [93]	2	$2^{n/2}$	2^n	1 [87]	
MJH [74]	2	$2^{n/2}$	2^n	1 [87]	
Jetchev et al. [61]	2	$2^{2n/3} \,..\, 2^n$	$2^n \,..\, 2^{3n/2}$	1 [87]	
Mennink [86]	3	2^n	$2^{3n/2}$	$2^{n/2}$ [87]	
MDC-4 [93]	4	$2^{5n/8} \,..\, 2^{3n/4}$ [68,88]	$2^{5n/4} \,..\, 2^{3n/2}$ [68,88]	$2^{n/4}$ [87]	

A more detailed security and efficiency comparison of some of these functions is presented by Bos et al. [30, App. A]. If the bound is expressed as $a \,..\, b$, this means that the security lower bound is a and the best known attack is b. A discussion of larger compression functions and for the case of iterated designs is given in Sect. 4

functions. All functions are shown to be differentiable from a random compression function by Kuwakado and Morii [69]. The results of [28,105] have been generalized in [29,46,120]. Gauravaram et al. [50] introduced the modified PGV constructions and proved that twelve of them achieve indifferentiability.

Compression functions of this form are known as single (block) length functions, meaning that the output size of the function equals the block length of the block cipher. However, when a collision security level of 128 bits or more is required, a larger compression function is needed and thus a block cipher with a block length of 256 bits or more; such block ciphers are not very common. This problem can be resolved by double (block) length hashing. Double length hashing is a well-established compression function design approach with $2n$-bit output based only on n-bit block ciphers.

We start by focusing on the simplest and most-studied type of double length compression functions, namely functions that compress $3n$ to $2n$ bits, and these can be divided into two classes: compression functions that internally evaluate a $2n$-bit keyed block cipher $E : \mathbb{Z}_2^{2n} \times \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ (which we will call the DBL^{2n} class), and ones that employ an n-bit keyed block cipher $E : \mathbb{Z}_2^n \times \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ (the DBL^n class). A state of the art is given in Table 1. These bounds also hold in the iteration, when a proper domain extender is applied (see Sect. 3).

4.1 Tight security of existing DBL^{2n} and DBL^n Designs

As indicated in Table 1, there are several remaining open problems. We highlight the gap for Jetchev et al.'s construction [61]: the scheme is proven collision secure up to $2^{2n/3}$ queries, but it is unknown whether this bound is tight, because no attack faster than the trivial one is known. Also for MDC-4, some non-trivial gaps remain, and it would be of interest to improve these gaps (we refer to Mennink [88] for a more detailed discussion of the state of the art of MDC-2 and MDC-4).

4.2 Existing DBL^{2n} and DBL^n designs in iteration

Table 1 focuses on compression function security, but when evaluated in a certain mode of operation (see Sect. 3) the state of the art is quite different.

We first consider collision and preimage resistance. While for DBL^{2n} the security gaps are comparable in the iterated setting, the DBL^n functions turn out to achieve much better security in the iteration. The main reason is that multiple DBL^n functions (such as MDC-2, MDC-4, and MJH) are designed to achieve security in the iteration (and not necessarily at compression function level). In more detail, the following results and gaps are present:

- The MDC-2 hash function is collision secure up to $2^{3n/5}$ queries [122] while an attack in $2^n/n$ queries exists [67]: this leaves a non-trivial gap $2^{3n/5} .. 2^n/n$. Regarding preimage resistance, [67] describes a tight attack on MDC-2 in 2^n queries;
- MDC-4 in a Merkle-Damgård mode of operation achieves collision resistance between $2^{5n/8} .. 2^n$ [88] and the preimage resistance between $2^{5n/4} .. 2^{3n/2}$ [60,88];
- MJH is collision resistant up to $2^n/n$ queries [74,75] and preimage resistant up to $2^n .. 2^{3n/2}$ queries [60];
- For Jetchev et al.'s construction, the collision gap remains in the iteration, and it is preimage resistant up to $2^n .. 2^{7n/4}$ queries;
- Similarly, Mennink's function is optimal secure in the iteration, but preimage resistant up to $2^{3n/2} .. 2^{7n/4}$ queries [59].

Next, for indifferentiability, the state of the art is completely different. While many functions are differentiable at the compression function level, they are likely secure in proper modes of operation. This is for instance already shown for Hirose's, Tandem-DM, and Abreast-DM [96]. For all other designs, such as Jetchev et al.'s, this question remains open. Also, the compression function indifferentiability result of Mennink's and MDC-4 composes well to hash function security, but for specific hash function designs, a better bound may be achievable if the indifferentiability result is derived for the hash function *directly*.

4.3 Optimality

Particularly in the DBL^n class, several fundamental design questions remain open. Firstly, is it possible to achieve optimal compression function security by only using two block cipher calls? Jetchev et al.'s design is a good step in the right direction (but only $2^{2n/3}$ security) and Mennink's makes three block cipher calls. The same question stands for the iterated scenario, where no design is proven optimally 2^n collision secure (see the previous section). Secondly, Mennink's design is relatively complicated, using three block ciphers in a non-parallelizable way, with constant finite field multiplications on the side. It remains an interesting open question to come up with more efficient designs with the same or improved security. Thirdly, is it possible to construct a design that is indifferentiable in more than $2^{n/2}$ queries, hence beyond the birthday bound of the block cipher?

4.4 Larger compression functions

We discussed single length functions, mapping $2n$ to n bits using an n-bit cipher, and double length functions, mapping $3n$ to $2n$ bits using an n-bit cipher. The next natural step would be to extend this practice to larger functions, and most importantly to a larger output size. Of interest here is a compression function design by Knudsen and Preneel [65] based on linear error-correcting codes $[r, k, d]$. The idea is to use r cn-to-n-bit random functions (where c is usually small) in parallel to obtain a kcn-to-rn-bit compression function, where the inputs to the underlying primitives are derived via the code. This compression function design has subsequently been analyzed by Watanabe [128], özen and Stam [100], Özen et al. [101] and Lee [71].

A generalization would be to consider the design of $(c+1)n$-to-cn-bit compression functions that achieve $2^{cn/2}$ security, still using n-bit block ciphers. Abed et al. [1] give a preliminary result in this direction, yet the construction uses an n-bit block cipher with a cn-bit key.

4.5 Beyond the ideal cipher model

So far, as explained in the beginning of this section, all results are in the ideal cipher model. In this case, the underlying cipher is assumed to be ideal and the security of the compression function is proven. These results, however, have to be considered with caution. Simon [118] already demonstrated that no provably collision resistant hash function exists based on just a one-way permutation, and it was demonstrated by Hirose [56] that the security results of Black et al. [28] do not hold beyond the ideal cipher model. In [26], Biryukov et al. present an attack on the Davies-Meyer compression function when instantiated with the AES [36] block cipher. A natural question to ask is how these results apply to double length hash functions.

An interesting approach in this direction is the model of ideal cipher irreducibility by Baecher et al. [13]. At a high level, it covers the idea of two constructions being equally secure for the same underlying blockcipher, which does not strictly need to be ideal.

5 Compression functions based on permutations

Block cipher based hash functions are characterized by the fact that the key input to the cipher depends on the input values; this implies that the key schedule has to be strong and that it needs to be executed for every encryption call, which entails a substantial computational cost. An alternative approach is to fix one or more keys, and restrict the hash function design to use the block cipher for these keys only. The usage of fixed-key block ciphers, or alternatively *permutations*, additionally offers the benefit that one does not need to implement an entire block cipher but only a limited number of instantiations of it.

This approach was introduced by Preneel et al. [104]. Black et al. [27] were the first to formally study these constructions, demonstrating that a $2n$-to-n-bit compression function f using one n-bit permutation π cannot be secure. This result has been generalized by Rogaway and Steinberger [114], and refined by Stam [119], Steinberger [123], and Steinberger, Sun, and Yang [124]. Consider any mn-to-rn-bit compression function using k n-bit permutations. Then, collisions can be found in at most $(2^n)^{1-(m-r+1)/(k+1)}$ queries to the underlying primitives. Collisions and preimages can even be found in at most $(2^n)^{1-(m-r/2)/k}$ and $(2^n)^{1-(m-r)/k}$ queries respectively, provided the compression function satisfies the "uniformity assumption" [114].

Most research in this area centered around $2n$-to-n-bit compression functions. Due to the above bounds, such a construction achieves optimal $2^{n/2}$ collision resistance *only if* it employs at least three permutations. Shrimpton and Stam [117] derived a compression function of this form, originally defined on one-way functions: $f(x_1, x_2) = f_1(x_1) \oplus f_3(f_1(x_1) \oplus f_2(x_2))$. This function is proven collision resistant up to $2^{n/2}$ queries, but preimages can be found after $2^{n/2}$ queries. An automated analysis of Rogaway and Steinberger [113] demonstrates that these results also hold if f_1, f_2, f_3 are defined as $f_i(x) = x \oplus \pi_i(x)$ for permutations π_1, π_2, π_3. Rogaway and Steinberger also proved a broad class of $2n$-to-n-bit compression functions using three distinct permutations and finite field scalar multiplications optimally collision and preimage secure (w.r.t. above bounds), and Mennink and Preneel [89] considered the general $2n$-to-n-bit compression functions based on three permutations. In general, this direction is well-understood.

5.1 Double length hashing

Similar to compression functions based on block ciphers (Sect. 4), the next and most logical step is $3n$-to-$2n$-bit compression functions using n-bit permutations. More and more, permutation based functions are used in a wide-pipe mode of operation, hence a permutation significantly larger than the output size is employed. For instance, in the SHA-3 finalists Grøstl [52], JH [129], and Keccak [21], permutations of size varying between 512 and 1600 bits are employed to construct a hash function with a 256-bit output. This makes the direction of double length hashing based on n-bit permutations not only a problem of theoretical, but foremost of practical interest.

Unfortunately, the above-mentioned bounds dictate collision security of at most $(2^n)^{1-2/(k+1)}$, only approaching 2^n in the limit. Nonetheless, it is a highly challenging research question to achieve security beyond the birthday-bound of the permutation, and already for a construction based on 4 permutations, it is an open problem to present a function that achieves $2^{3n/5}$ collision security (this open problem was already posed by Stam in 2008 [119]).

5.2 Indifferentiability

Unlike the case of block cipher based compression functions (Sect. 4), not much is known for the indifferentiability of permutation based compression functions. A interesting exception is the function $f(x) = \pi(x) \oplus \pi^{-1}(x)$ [44], which achieves $2^{n/2}$ security. Mandal et al. [82] proved that the XOR of two permutations, $f(x) = \pi_1(x) \oplus \pi_2(x)$, achieves $2^{2n/3}$ indifferentiability (in [90], Mennink and Preneel pointed out a subtle flaw in that proof but were able to restore the original security claim). These functions are both non-compressing. Yet, it would be interesting to determine which indifferentiability guarantees the compression functions of [89,113,117] offer: by composition such guarantees immediately carry over to full hash function designs. We point out the related direction of preimage-awareness [45]: in this work (among others) the compression function of Shrimpton and Stam is proven preimage-aware, and under certain circumstances this result propagates to an indifferentiable hash function.

5.3 Sum-capture problem

The result of Mennink and Preneel [89] relies on a conjecture, that (informally) bounds the number of solutions to $a \oplus b = c$ for $(a, b, c) \in A \times B \times C$, where C is a set of random elements and A and B are two arbitrarily chosen sets of the same size as C. This conjecture was used to bound the number of solutions $x_1 \oplus x_2 = x_3 \oplus \pi(x_3)$ where x_1, x_2, x_3 can be selected by the adversary and π is a random permutation. A part of this conjecture has been proven by Steinberger [121] (reconfirming a result of Babai [10]). The problem and variations thereof found later adoption in, among others, digital signatures [64] and the security of Even-Mansour [34]. It is therefore of interest to further analyze the conjecture and to improve the existing results.

6 Hash functions based on permutations

Since the introduction of sponge functions [22], an active research area is to construct from a permutation a secure hash function, without requiring a secure compression function. It has been a standard practice in the cryptographic community to call hash functions "sponge-like" if they bear resemblances with the original sponge design in terms of iterating a wide state and employing underlying permutations in an extraction and absorbing phases. Examples of such functions include Grindahl [66] and the second round SHA-3 candidate hash functions CubeHash [17], Fugue [129], JH [21], and Keccak [54]. Sponge and sponge-like functions have been subsequently generalized in the form of parazoa functions [4] and the broadened Stam hash function design [32]. The composition claims of previous sections then do not work, and security of the hash function designs has to be derived directly. In this section, we discuss some interesting open problems regarding the construction of a hash function $\mathcal{H}$ from a permutation π, where the compression function (based on π) need not be secure. Throughout, π is assumed to be an ideal permutation.

6.1 Generalizations

Indifferentiability proofs are nowadays still design-specific, even if all of these proofs have some characteristics in common. On the other hand, generalized indifferentiability conventionally results in worse bounds; this happened for the parazoa functions and the broadened Stam hash function design. An interesting problem is to re-consider the problem of generalizing sponge functions. The target would be to generalize them in a possibly different fashion, in such a way that a smaller set of designs is covered but an improved bound can be obtained.

6.2 Preimage security

Writing s to be the state size of sponge functions, and r the rate (the number of message bits compressed per round), sponge functions are proven indifferentiable up to $2^{(s-r)/2}$ queries. Putting $s = 2n + r$, where n is the hash function output size, this renders indifferentiability up to 2^n queries. This moreover implies that such sponge function is collision secure up to $2^{n/2}$ and (second) preimage secure up to 2^n queries.

On the other hand, functions such as JH-512 are only proven to achieve 2^{256} indifferentiability [24,95] and collision security [72]. In fact, JH-512 is proven preimage secure up to 2^{256} queries only, and it still remains an open problem to prove security beyond this bound (the best known attack requires approximately $2^{512}/512$ queries [24]). This is also mentioned as an open problem in the SHA-3 classification of Andreeva et al. [5]. A similar remark applies to generalized designs. For instance, for specific parameter choices the same issue appears for parazoa functions. It would be interesting to determine the preimage security of parazoa functions beyond the indifferentiability bound.

6.3 SHA-3

On October 2nd, 2012, NIST announced that Keccak has been selected as the winner of the SHA-3 competition. Keccak is about to be standardized and will likely become the most prominent hash function in the next decennia. It is in the line of expectation that it will not only be used as a "simple" hash function, but that it will fulfill different purposes too. Several extensions and variant modes of operation for Keccak have already appeared [19,20,23]. Additionally, the ongoing CAESAR competition for the design of a new authenticated encryption scheme [31] received various designs that were in some way based on the sponge in general or SHA-3 in particular. This area still leaves many open problems that are beyond the scope of this paper.

7 Conclusions

We summarized the state of the art on generic hash function security, considering the design of a hash function based on a compression function as well as the design of a compression function from one or more block ciphers of permutations. In each of these directions, we pointed out relevant open problems.

The study of open problems in cryptography is highly relevant. Both the theoretical and practical approaches result in better security bounds and guarantees and in a better understanding of what is achievable and what isn't. It also allows for the timely identification of problems and may preclude misuses.

As a final remark, we note that some proofs in literature are quite generic. These proofs conventionally consist of dividing a hard problem into multiple smaller problems. These smaller problems often show similarities, and automated proof techniques will likely contribute to an easier and more convincing validation of proofs. Rogaway and Steinberger [113] developed an automated proof technique to analyze permutation based compression functions, and similarly, automated techniques have been developed for the indifferentiability of the Merkle-Damgård hash function research by Backes et al. [11] and Daubignard et al. [38]. Indifferentiability proofs for hash functions will definitely benefit from automated techniques because they tend to be intricate and difficult to verify.

Acknowledgments This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007). Elena Andreeva and Bart Mennink are Postdoctoral Fellows of the Research Foundation—Flanders (FWO).

References

1. Abed F., Forler C., List E., Lucks S., Wenzel J.: Counter-bDM: a provably secure family of multi-block-length compression functions. In: Progress in Cryptology—AFRICACRYPT 2014. Lecture Notes in Computer Science, vol. 8469, pp. 440–458. Springer, Heidelberg (2014).
2. Andreeva E., Mennink B., Preneel B.: Security properties of domain extenders for cryptographic hash functions. J. Inf. Process. Syst. **6**(4), 453–480 (2010).
3. Andreeva E., Mennink B., Preneel B.: Security reductions of the second round SHA-3 candidates. In: Information Security Conference—ISC 2010. Lecture Notes in Computer Science, vol. 6531, pp. 39–53. Springer, Heidelberg (2010).
4. Andreeva E., Mennink B., Preneel B.: The parazoa family: generalizing the sponge hash functions. Int. J. Inf. Secur. **11**(3), 149–165 (2012).
5. Andreeva E., Mennink B., Preneel B., Škrobot M.: Security analysis and comparison of the SHA-3 Finalists BLAKE. In: Grøstl J.H., Keccak and Skein (eds.) Progress in Cryptology—AFRICACRYPT 2012. Lecture Notes in Computer Science, vol. 7374, pp. 287–305. Springer, Heidelberg (2012).
6. Andreeva E., Neven G., Preneel B., Shrimpton T.: Seven-property-preserving iterated hashing: ROX. In: Advances in Cryptology—ASIACRYPT 2007. Lecture Notes in Computer Science, vol. 4833, pp. 130–146. Springer, Heidelberg (2007).
7. Andreeva E., Preneel B.: A three-property-secure hash function. In: Selected Areas in Cryptography 2008. Lecture Notes in Computer Science, vol. 5381, pp. 228–244. Springer, Heidelberg (2008).
8. Andreeva E., Stam M.: The symbiosis between collision and preimage resistance. In: IMA International Conference 2011. Lecture Notes in Computer Science, vol. 7089, pp. 152–171. Springer, Heidelberg (2011).

9. Armknecht F., Fleischmann E., Krause M., Lee J., Stam M., Steinberger J.P.: The preimage security of double-block-length compression functions. In: Advances in Cryptology—ASIACRYPT 2011. Lecture Notes in Computer Science, vol. 7073, pp. 233–251. Springer, Heidelberg (2011).
10. Babai L.: The Fourier Transform and Equations over Finite Abelian Groups (Lecture Notes, version 1.3) (2002). http://people.cs.uchicago.edu/laci/reu02/fourier.pdf.
11. Backes M., Barthe G., Berg M., Grégoire B., Kunz C., Skoruppa M., Béguelin S.Z.: Verified security of Merkle-Damgård. In: Computer Security Foundations Symposium—CSF 2012. pp. 354–368. IEEE Comput. Soc. (2012).
12. Baecher P., Brzuska C., Mittelbach A.: Reset indifferentiability and its consequences. In: Advances in Cryptology—ASIACRYPT 2013 (I). Lecture Notes in Computer Science, vol. 8269, pp. 154–173. Springer, Heidelberg (2013).
13. Baecher P., Farshim P., Fischlin M., Stam M.: Ideal-cipher (ir)reducibility for blockcipher-based hash functions. In: Advances in Cryptology—EUROCRYPT 2013. Lecture Notes in Computer Science, vol. 7881, pp. 426–443. Springer, Heidelberg (2013).
14. Barreto P., Rijmen V.: The WHIRLPOOL hashing function (2003). http://www.larc.usp.br/pbarreto/whirlpool.zip.
15. Bellare M., Canetti R., Krawczyk H.: A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In: Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing. pp. 419–428. ACM (1998).
16. Bellare M., Rogaway P.: Collision-resistant hashing: towards making UOWHFs practical. In: Advances in Cryptology—CRYPTO '97. Lecture Notes in Computer Science, vol. 1294, pp. 470–484. Springer, Heidelberg (1997).
17. Bernstein D.: CubeHash specificatio. Submission to NIST's SHA-3 Competition (2009).
18. Bertoni G., Daemen J., Peeters M., Assche G.: Sufficient conditions for sound tree and sequential hashing modes. Cryptol. ePrint Arch. Report 2009/210 (2009).
19. Bertoni G., Daemen J., Peeters M., Assche G.: Duplexing the sponge: single-pass authenticated encryption and other applications. In: Selected Areas in Cryptography 2011. Lecture Notes in Computer Science, vol. 7118, pp. 320–337. Springer, Heidelberg (2011).
20. Bertoni G., Daemen J., Peeters M., Assche G.: On the security of the keyed sponge construction. In: Symmetric Key Encryption Workshop (SKEW 2011) (2011).
21. Bertoni G., Daemen J., Peeters M., Assche G.: The KECCAK sponge function family, submission to NIST's SHA-3 competition (2011).
22. Bertoni G., Daemen J., Peeters M., Assche G.: Sponge functions (ECRYPT Hash Function Workshop 2007).
23. Bertoni G., Daemen J., Peeters M., Van Assche G.: Sakura: a flexible coding for tree hashing. In: Applied Cryptography and Network Security—ACNS 2014. Lecture Notes in Computer Science, vol. 8479, pp. 217–234. Springer, Heidelberg (2014).
24. Bhattacharyya R., Mandal A., Nandi M.: Security analysis of the mode of JH hash function. In: Fast Software Encryption 2010. Lecture Notes in Computer Science, vol. 6147, pp. 168–191. Springer, Heidelberg (2010).
25. Biham E., Dunkelman O.: A framework for iterative hash functions—HAIFA. Cryptology ePrint Archive, Report 2007/278 (2007).
26. Biryukov A., Khovratovich D., Nikolić I.: Distinguisher and related-key attack on the full AES-256. In: Advances in Cryptology—CRYPTO 2009. Lecture Notes in Computer Science, vol. 5677, pp. 231–249. Springer, Heidelberg (2009).
27. Black J., Cochran M., Shrimpton T.: On the impossibility of highly-efficient blockcipher-based hash functions. In: Advances in Cryptology—EUROCRYPT 2005. Lecture Notes in Computer Science, vol. 3494, pp. 526–541. Springer, Heidelberg (2005).
28. Black J., Rogaway P., Shrimpton T.: Black-box analysis of the block-cipher-based hash-function constructions from PGV. In: Advances in Cryptology—CRYPTO 2002. Lecture Notes in Computer Science, vol. 2442, pp. 320–335. Springer, Heidelberg (2002).
29. Black J., Rogaway P., Shrimpton T., Stam M.: An analysis of the blockcipher-based hash functions from PGV. J. Cryptol. **23**(4), 519–545 (2010).
30. Bos J., Özen O., Stam M.: Efficient hashing using the AES instruction set. In: Cryptographic Hardware and Embedded Systems—CHES 2011. Lecture Notes in Computer Science, vol. 6917, pp. 507–522. Springer, Heidelberg (2011).
31. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness (2014). http://competitions.cr.yp.to/caesar.html.

32. Canteaut A., Fuhr T., Naya-Plasencia M., Paillier P., Reinhard J.R., Videau M.: A unified indifferentiability proof for permutation-or block cipher-based hash functions. Cryptology ePrint Archive, Report 2012/363 (2012).
33. Chang D., Lee S., Nandi M., Yung M.: Indifferentiable security analysis of popular hash functions with prefix-free padding. In: Advances in Cryptology—ASIACRYPT 2006. Lecture Notes in Computer Science, vol. 4284, pp. 283–298. Springer, Heidelberg (2006).
34. Chen S., Lampe R., Lee J., Seurin Y., Steinberger J.P.: Minimizing the two-round Even-Mansour cipher. In: Advances in Cryptology—CRYPTO 2014 (I). Lecture Notes in Computer Science, vol. 8616, pp. 39–56. Springer, Heidelberg (2014).
35. Coron J., Dodis Y., Malinaud C., Puniya P.: Merkle-Damgård revisited: how to construct a hash function. In: Advances in Cryptology—CRYPTO 2005. Lecture Notes in Computer Science, vol. 3621, pp. 430–448. Springer, Heidelberg (2005).
36. Daemen J., Rijmen V.: The Design of Rijndael: AES—The Advanced Encryption Standard. Springer, Berlin (2002).
37. Damgård I.: A design principle for hash functions. In: Advances in Cryptology—CRYPTO '89. Lecture Notes in Computer Science, vol. 435, pp. 416–427. Springer, Heidelberg (1990).
38. Daubignard M., Lafourcade P., Lakhnech Y.: Generic indifferentiability proofs of hash designs. In: Computer Security Foundations Symposium—CSF 2012. Lecture Notes in Computer Science, IEEE Computer Society, Washington, pp. 340–353 (2012).
39. Davies, D.W., Price, W.L.: Security for Computer Networks—an Introduction to Data Security in Teleprocessing and Electronic Funds Transfer (2. ed.). Wiley Series in Communication and Distributed Systems (1989).
40. Dean R.: Formal Aspects of Mobile Code Security. Ph.D. thesis, Princeton University, Princeton (1999).
41. Demay G., Gaži P., Hirt M., Maurer U.: Resource-restricted indifferentiability. In: Advances in Cryptology—EUROCRYPT 2013. Lecture Notes in Computer Science, vol. 7881, pp. 664–683. Springer, Heidelberg (2013).
42. Diffie W., Hellman M.: New directions in cryptography. IEEE Trans. Inf. Theory **22**(6), 655–654 (1976).
43. Dobbertin H., Bosselaers A., Preneel B.: RIPEMD-160: a strengthened version of RIPEMD. In: Fast Software Encryption '96. Lecture Notes in Computer Science, vol. 1039, pp. 71–82. Springer, Heidelberg (1996).
44. Dodis Y., Pietrzak K., Puniya P.: A new mode of operation for block ciphers and length-preserving MACs. In: Advances in Cryptology—EUROCRYPT 2008. Lecture Notes in Computer Science, vol. 4965, pp. 198–219. Springer, Heidelberg (2008).
45. Dodis Y., Ristenpart T., Shrimpton T.: Salvaging Merkle-Damgård for practical applications. In: Advances in Cryptology—EUROCRYPT 2009. Lecture Notes in Computer Science, vol. 5479, pp. 371–388. Springer, Heidelberg (2009).
46. Duo L., Li C.: Improved collision and preimage resistance bounds on PGV schemes. Cryptology ePrint Archive, Report 2006/462 (2006).
47. Ferguson N., Lucks S., Schneier B., Whiting D., Bellare M., Kohno T., Callas J., Walker J.: The Skein Hash Function Family, submission to NIST's SHA-3 competition (2009).
48. Fleischmann E., Gorski M., Lucks S.: Security of cyclic double block length hash functions. In: IMA International Conference 2009. Lecture Notes in Computer Science, vol. 5921, pp. 153–175. Springer, Heidelberg (2009).
49. Gauravaram P.: Security analysis of salt||password hashes. In: Advanced Computer Science Applications and Technologies—ACSAT 2012, pp. 25–30. IEEE (2012).
50. Gauravaram P., Bagheri N., Knudsen L.R.: Building indifferentiable compression functions from the PGV compression functions. Des. Codes Cryptogr. doi:10.1007/s10623-014-0020-z (2014).
51. Gauravaram P., Knudsen L.R.: Security analysis of randomize-hash-then-sign digital signatures. J. Cryptol. **25**(4), 748–779 (2012).
52. Gauravaram P., Knudsen L.R., Matusiewicz K., Mendel F., Rechberger C., Schläffer M., Thomsen S.: Grøstl—a SHA-3 candidate. Submitted to NIST's SHA-3 Competition (2011).
53. Gong Z., Lai X., Chen K.: A synthetic indifferentiability analysis of some block-cipher-based hash functions. Des. Codes Cryptogr. **48**(3), 293–305 (2008).
54. Halevi S., Hall W., Jutla C.: The hash function "Fugue". Submitted to NIST's SHA-3 Competition (2009).
55. Halevi S., Krawczyk H.: Strengthening digital signatures via randomized hashing. In: Advances in Cryptology—CRYPTO 2006. Lecture Notes in Computer Science, vol. 4117, pp. 41–59. Springer, Heidelberg (2006).

56. Hirose S.: Secure block ciphers are not sufficient for one-way hash functions in the preneel-govaerts-vandewalle mode. In: Selected Areas in Cryptography 2002. Lecture Notes in Computer Science, vol. 2595, pp. 339–352. Springer, Heidelberg (2003).
57. Hirose S.: Provably secure double-block-length hash functions in a black-box model. In: Information Security and Cryptology—ICISC 2004. Lecture Notes in Computer Science, vol. 3506, pp. 330–342. Springer, Heidelberg (2005).
58. Hirose S.: Some plausible constructions of double-block-length hash functions. In: Fast Software Encryption 2006. Lecture Notes in Computer Science, vol. 4047, pp. 210–225. Springer, Heidelberg (2006).
59. Hong D., Kwon D.: Cryptanalysis of some double-block-length hash modes of block ciphers with n-bit block and n-bit key. Cryptology ePrint Archive Report 2013/174 (2013).
60. Hong D., Kwon D.: Cryptanalysis of double-block-length hash modes MDC-4 and MJH. IEICE Trans. 97-A(8), 1747–1753 (2014).
61. Jetchev D., Özen O., Stam M.: Collisions are not incidental: a compression function exploiting discrete geometry. In: Theory of Cryptography Conference 2012. Lecture Notes in Computer Science, vol. 7194, pp. 303–320. Springer, Heidelberg (2012).
62. Kelsey J., Kohno T.: Herding hash functions and the Nostradamus attack. In: Advances in Cryptology—EUROCRYPT 2006. Lecture Notes in Computer Science, vol. 4004, pp. 183–200. Springer, Heidelberg (2006).
63. Kelsey J., Schneier B.: Second preimages on n-bit hash functions for much less than 2^n work. In: Advances in Cryptology—EUROCRYPT 2005. Lecture Notes in Computer Science, vol. 3494, pp. 474–490. Springer, Heidelberg (2005).
64. Kiltz E., Pietrzak K., Szegedy M.: Digital signatures with minimal overhead from indifferentiable random invertible functions. In: Advances in Cryptology—CRYPTO 2013 (I). Lecture Notes in Computer Science, vol. 8042, pp. 571–588. Springer, Heidelberg (2013).
65. Knudsen L.R., Preneel B.: Fast and secure hashing based on codes. In: Advances in Cryptology—CRYPTO '97. Lecture Notes in Computer Science, vol. 1294, pp. 485–498. Springer, Heidelberg (1997).
66. Knudsen L.R., Preneel B.: Construction of secure and fast hash functions using nonbinary error-correcting codes. IEEE Trans. Inf. Theory **48**(9), 2524–2539 (2002).
67. Knudsen L.R., Rechberger C., Thomsen S.: The Grindahl hash functions. In: Fast Software Encryption 2007. Lecture Notes in Computer Science, vol. 4593, pp. 39–57. Springer, Heidelberg (2007).
68. Knudsen L.R., Mendel F., Rechberger C., Thomsen S.: Cryptanalysis of MDC-2. In: Advances in Cryptology—EUROCRYPT 2009. Lecture Notes in Computer Science, vol. 5479, pp. 106–120. Springer, Heidelberg (2009).
69. Kuwakado H., Morii M.: Indifferentiability of single-block-length and rate-1 compression functions. IEICE Trans. 90-A(10), 2301–2308 (2007).
70. Lai X., Massey J.: Hash function based on block ciphers. In: Advances in Cryptology—EUROCRYPT '92. Lecture Notes in Computer Science, vol. 658, pp. 55–70. Springer, Heidelberg (1992).
71. Lee J.: Provable security of the Knudsen-Preneel compression functions. In: Advances in Cryptology—ASIACRYPT 2012. Lecture Notes in Computer Science, vol. 7658, pp. 504–525. Springer, Heidelberg (2012).
72. Lee J., Hong D.: Collision resistance of the JH hash function. IEEE Trans. Inf. Theory **58**(3), 1992–1995 (2012).
73. Lee J., Kwon D.: The security of abreast-DM in the ideal cipher model. IEICE Trans. 94-A(1), 104–109 (2011).
74. Lee J., Stam M.: MJH: a faster alternative to MDC-2. In: CT-RSA 2011. Lecture Notes in Computer Science, vol. 6558, pp. 213–236. Springer, Heidelberg (2011).
75. Lee J., Stam M.: MJH: a faster alternative to MDC-2. Des. Codes Cryptogr. doi:10.1007/978-3-642-19074-2_15 (2014).
76. Lee J., Stam M., Steinberger J.P.: The collision security of Tandem-DM in the ideal cipher model. In: Advances in Cryptology—CRYPTO 2011. Lecture Notes in Computer Science, vol. 6841, pp. 561–577. Springer, Heidelberg (2011).
77. Lee J., Stam M., Steinberger J.P.: The preimage security of double-block-length compression functions. Cryptol. ePrint Arch. Report 2011/210 (2011).
78. Lee W., Chang D., Lee S., Sung S., Nandi M.: New parallel domain extenders for UOWHF. In: Advances in Cryptology—ASIACRYPT 2003. Lecture Notes in Computer Science, vol. 2894, pp. 208–227. Springer, Heidelberg (2003).
79. Lee W., Chang D., Lee S., Sung S., Nandi M.: Construction of UOWHF: two new parallel methods. IEICE Trans. **88-A**(1), 49–58 (2005).

80. Luo Y., Gong Z., Duan M., Zhu B., Lai X.: Revisiting the indifferentiability of PGV hash functions. Cryptology ePrint Archive Report 2009/265 (2009).
81. Luykx A., Andreeva E., Mennink B., Preneel B.: Impossibility results for indifferentiability with resets. Cryptology ePrint Archive Report 2012/644 (2012).
82. Mandal A., Patarin J., Nachef V.: Indifferentiability beyond the birthday bound for the xor of two public random permutations. In: Progress in Cryptology—INDOCRYPT 2010. Lecture Notes in Computer Science, vol. 6498, pp. 69–81. Springer, Heidelberg (2010).
83. Matyas S., Meyer C., Oseas J.: Generating strong one-way functions with cryptographic algorithm. IBM Tech. Discl. Bull. **27**(10A), 5658–5659 (1985).
84. Maurer U., Renner R., Holenstein C.: Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In: Theory of Cryptography Conference 2004. Lecture Notes in Computer Science, vol. 2951, pp. 21–39. Springer, Heidelberg (2004).
85. Menezes A., Oorschot P., Vanstone S.: Handbook of Applied Cryptography. CRC Press, New York (1997).
86. Mennink B.: Optimal collision security in double block length hashing with single length key. In: Advances in Cryptology—ASIACRYPT 2012. Lecture Notes in Computer Science, vol. 7658, pp. 526–543. Springer, Heidelberg (2012).
87. Mennink B.: Indifferentiability of double length compression functions. In: IMA International Conference 2013. Lecture Notes in Computer Science, vol. 8308, pp. 232–251. Springer, Heidelberg (2013).
88. Mennink B.: On the collision and preimage security of MDC-4 in the ideal cipher model. Des. Codes Cryptogr. **73**(1), 121–150 (2014).
89. Mennink B., Preneel B.: Hash functions based on three permutations: a generic security analysis. In: Advances in Cryptology—CRYPTO 2012. Lecture Notes in Computer Science, vol. 7417, pp. 330–347. Springer, Heidelberg (2012).
90. Mennink B., Preneel B.: On the XOR of multiple random permutations. In: Applied Cryptography and Network Security - ACNS 2015. Lecture Notes in Computer Science. Springer, Heidelberg (2015), to appear.
91. Merkle R.: One way hash functions and DES. In: Advances in Cryptology—CRYPTO ’89. Lecture Notes in Computer Science, vol. 435, pp. 428–446. Springer, Heidelberg (1990).
92. Merkle R.C.: Secrecy, Authentication and Public Key Systems. Ph.D. thesis, UMI Research Press, Italy (1979).
93. Meyer C., Schilling M.: Secure program load with manipulation detection code. In: Proceedings of Securicom. pp. 111–130 (1988).
94. Miyaguchi S., Ohta K., Iwata M.: Confirmation that some hash functions are not collision free. In: Advances in Cryptology—EUROCRYPT ’90. Lecture Notes in Computer Science, vol. 473, pp. 326–343. Springer, Heidelberg (1990).
95. Moody D., Paul S., Smith-Tone D.: Improved indifferentiability security bound for the JH mode. Cryptology ePrint Archive Report 2012/278 (2012).
96. Naito Y.: Blockcipher-based double-length hash functions for pseudorandom oracles. In: Selected Areas in Cryptography 2011. Lecture Notes in Computer Science, vol. 7118, pp. 338–355. Springer, Heidelberg (2011).
97. Naito Y., Yoneyama K., Ohta K.: Reset indifferentiability from weakened random oracle salvages one-pass hash functions. In: Applied Cryptography and Network Security—ACNS 2014. Lecture Notes in Computer Science, vol. 8479, pp. 235–252. Springer, Heidelberg (2014).
98. National Institute for Standards and Technology: Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA3) family (2007).
99. Özen O., Stam M.: Another glance at double-length hashing. In: IMA International Conference 2009. Lecture Notes in Computer Science, vol. 5921, pp. 176–201. Springer, Heidelberg (2009).
100. Özen O., Stam M.: Collision attacks against the Knudsen-Preneel compression functions. In: Advances in Cryptology—ASIACRYPT 2010. Lecture Notes in Computer Science, vol. 6477, pp. 76–93. Springer, Heidelberg (2010).
101. Özen O., Shrimpton T., Stam M.: Attacking the Knudsen-Preneel compression functions. In: Fast Software Encryption 2010. Lecture Notes in Computer Science, vol. 6147, pp. 94–115. Springer, Heidelberg (2010).
102. Password Hashing Competition (2013). https://password-hashing.net/.
103. Preneel B., van Oorschot P.C.: On the security of iterated message authentication codes. IEEE Trans. Inf. Theory **45**(1), 188–199 (1999).
104. Preneel B., Govaerts R., Vandewalle J.: On the power of memory in the design of collision resistant hash functions. In: Advances in Cryptology—AUSCRYPT ’92. Lecture Notes in Computer Science, vol. 718, pp. 105–121. Springer, Heidelberg (1992).

105. Preneel B., Govaerts R., Vandewalle J.: Hash functions based on block ciphers: a synthetic approach. In: Advances in Cryptology—CRYPTO '93. Lecture Notes in Computer Science, vol. 773, pp. 368–378. Springer, Heidelberg (1993).
106. Reyhanitabar M., Susilo W., Mu Y.: Enhanced security notions for dedicated-key hash functions: definitions and relationships. In: Fast Software Encryption 2010. Lecture Notes in Computer Science, vol. 6147, pp. 192–211. Springer, Heidelberg (2010).
107. Ristenpart T., Shacham H., Shrimpton T.: Careful with composition: limitations of the indifferentiability framework. In: Advances in Cryptology—EUROCRYPT 2011. Lecture Notes in Computer Science, vol. 6632, pp. 487–506. Springer, Heidelberg (2011).
108. Rivest R.: The MD4 message digest algorithm. In: Advances in Cryptology—CRYPTO '90. Lecture Notes in Computer Science, vol. 537, pp. 303–311. Springer, Heidelberg (1991).
109. Rivest R.: The MD5 message-digest algorithm. Request for Comments (RFC) 1321 (1992).
110. Rivest R.: Abelian square-free dithering for iterated hash functions. In: ECRYPT Hash Function Workshop 2005.
111. Rivest R.L., Agre B., Bailey D.V., Crutchfield C., Dodis Y., Elliott K., Khan F.A., Krishnamurthy J., Lin Y., Reyzin L., Shen E., Sukha J., Sutherland D., Tromer E., Yin Y.L.: The MD6 hash function. Submitted to NIST's SHA-3 Competition (2008).
112. Rogaway P., Shrimpton T.: Cryptographic hash-function basics: definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In: Fast software encryption 2004. Lecture Notes in Computer Science, vol. 3017, pp. 371–388. Springer, Heidelberg (2004).
113. Rogaway P., Steinberger J.P.: Constructing cryptographic hash functions from fixed-key blockciphers. In: Advances in Cryptology—CRYPTO 2008. Lecture Notes in Computer Science, vol. 5157, pp. 433–450. Springer, Heidelberg (2008).
114. Rogaway P., Steinberger J.P.: Security/efficiency tradeoffs for permutation-based hashing. In: Advances in Cryptology—EUROCRYPT 2008. Lecture Notes in Computer Science, vol. 4965, pp. 220–236. Springer, Heidelberg (2008).
115. Sarkar P.: Construction of universal one-way hash functions: tree hashing revisited. Discret. Appl. Math. **155**(16), 2174–2180 (2007).
116. National Institute of Standards and Technology. Secure Hash Standard (SHS). Federal Information Processing Standards Publication 180–3 (2008).
117. Shrimpton T., Stam M.: Building a collision-resistant compression function from non-compressing primitives. In: International Colloquium on Automata, Languages and Programming—ICALP (2) 2008. Lecture Notes in Computer Science, vol. 5126, pp. 643–654. Springer, Heidelberg (2008).
118. Simon D.: Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In: Advances in Cryptology—EUROCRYPT '98. Lecture Notes in Computer Science, vol. 1403, pp. 334–345. Springer, Heidelberg (1998).
119. Stam M.: Beyond uniformity: better security/efficiency tradeoffs for compression functions. In: Advances in Cryptology—CRYPTO 2008. Lecture Notes in Computer Science, vol. 5157, pp. 397–412. Springer, Heidelberg (2008).
120. Stam M.: Blockcipher-based hashing revisited. In: Fast Software Encryption 2009. Lecture Notes in Computer Science, vol. 5665, pp. 67–83. Springer, Heidelberg (2009).
121. Steinberger J.: The Sum-Capture Problem for Abelian Groups (2014). http://arxiv.org/abs/1309.5582.
122. Steinberger J.P.: The collision intractability of MDC-2 in the ideal-cipher model. In: Advances in Cryptology—EUROCRYPT 2007. Lecture Notes in Computer Science, vol. 4515, pp. 34–51. Springer, Heidelberg (2007).
123. Steinberger J.P.: Stam's collision resistance conjecture. In: Advances in Cryptology—EUROCRYPT 2010. Lecture Notes in Computer Science, vol. 6110, pp. 597–615. Springer, Heidelberg (2010).
124. Steinberger J.P., Sun X., Yang Z.: Stam's conjecture and threshold phenomena in collision resistance. In: Advances in Cryptology—CRYPTO 2012. Lecture Notes in Computer Science, vol. 7417, pp. 384–405. Springer, Heidelberg (2012).
125. Tsudik G.: Message authentication with one-way hash functions. In: ACM Comput. Commun. Rev. pp. 29–38. ACM, New York (1992).
126. Wang X., Yu H.: How to break MD5 and other hash functions. In: Advances in Cryptology—EUROCRYPT 2005. Lecture Notes in Computer Science, vol. 3494, pp. 19–35. Springer, Heidelberg (2005).
127. Wang X., Yin Y., Yu H.: Finding collisions in the full SHA-1. In: Advances in Cryptology—CRYPTO 2005. Lecture Notes in Computer Science, vol. 3621, pp. 17–36. Springer, Heidelberg (2005).
128. Watanabe D.: A note on the security proof of Knudsen-Preneel construction of a hash function (2006).
129. Wu H.: The Hash Function JH. Submitted to NIST's SHA-3 Competition (2011).

130. Yasuda K.: How to fill up Merkle-Damgård hash functions. In: Advances in Cryptology—ASIACRYPT 2008. Lecture Notes in Computer Science, vol. 5350, pp. 272–289. Springer, Heidelberg (2008).
131. Yuval G.: How to swindle Rabin. Cryptologia **3**(3), 187–191 (1979).
132. Zheng Y., Pieprzyk J., Seberry J.: HAVAL—a one-way hashing algorithm with variable length of output. In: Advances in Cryptology—AUSCRYPT '92. Lecture Notes in Computer Science, vol. 718, pp. 83–104. Springer, Heidelberg (1993).