3C -
A Provably Secure Pseudorandom Function and Message Authentication Code.
A New mode of operation for Cryptographic Hash Function *

Praveen Gauravaram[1] **, William Millan[1], Juanma Gonzalez Neito[1], Edward Dawson[1]

Information Security Institute (ISI), QUT, Australia.
p.gauravaram@isi.qut.edu.au,{b.millan,j.gonzaleznieto,e.dawson}@qut.edu.au

Abstract. We propose a new cryptographic construction called **3C**, which works as a pseudorandom function (PRF), message authentication code (MAC) and cryptographic hash function. The **3C**-construction is obtained by modifying the Merkle-Damgård iterated construction used to construct iterated hash functions. We assume that the compression functions of Merkle-Damgård iterated construction realize a family of fixed-length-input pseudorandom functions (FI-PRFs). A concrete security analysis for the family of **3C**- variable-length-input pseudorandom functions (VI-PRFs) is provided in a precise and quantitative manner. The **3C**- VI-PRF is then used to realize the **3C**- MAC construction called one-key NMAC (O-NMAC). O-NMAC is a more efficient variant of NMAC and HMAC in the applications where key changes frequently and the key cannot be cached. The **3C**-construction works as a new mode of hash function operation for the hash functions based on Merkle-Damgård construction such as MD5 and SHA-1. The generic **3C**- hash function is more resistant against the recent differential multi-block collision attacks than the Merkle-Damgård hash functions and the extension attacks do not work on the **3C**- hash function. The **3C-X** hash function is the simplest and efficient variant of the generic **3C** hash function and it is the simplest modification to the Merkle-Damgård hash function that one can achieve. We provide the security analysis for the functions **3C** and **3C-X** against multi-block collision attacks and generic attacks on hash functions. We combine the wide-pipe hash function with the **3C** hash function for even better security against some generic attacks and differential attacks. The **3C**-construction has all these features at the expense of one extra iteration of the compression function over the Merkle-Damgård construction.

1 Introduction

In 1989, Damgård [9] and Merkle [20] independently proposed a similar iterative structure to construct collision resistant cryptographic hash functions. Since then, this design has been called Merkle-Damgård (MD) iterative structure. The design principle of this construction is

> "If there exists a computationally collision free function f from m bits to t bits where $m > t$, then there exists a computationally collision free function h mapping messages of arbitrary polynomial lengths to t-bit strings." [9]

It has been believed that the problem of designing a collision resistant hash function reduces to the problem of designing a collision resistant compression function. It is known that, a compression function secure against the fixed initial value (IV) collisions is necessary but not sufficient to generate a secure hash function [19]. The multi-block differential collision attacks on the hash

* This is a draft version of the work in progress. Any comments, suggestions and corrections via email are very much appreciated and acknowledged in the final draft.

** Author (http://www.isrc.qut.edu.au/people/subramap) has been supported by the QUT-PRA and ISI top-up funding

functions MD5, SHA-0 and SHA-1 [7,28,29] proves this insufficiency. These attacks show that these iterated hash functions do *not properly preserve* the collision resistance property of the compression function with its fixed IV. For details on the background information see [11].

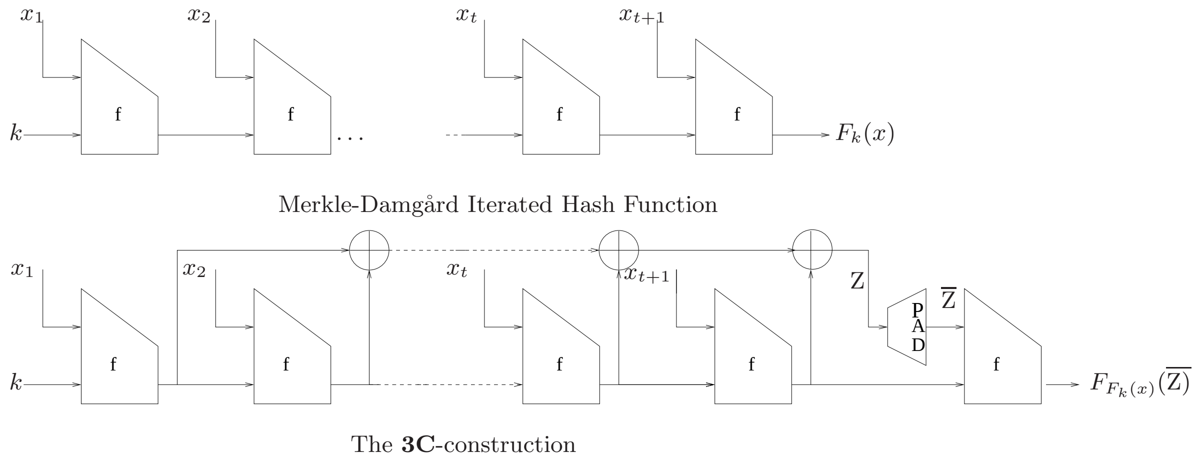

Fig. 1. The views of MD and **3C**-construction

In this work, we propose a new variant to the Merkle-Damgård iterative construction called **3C**-construction. The keyed versions of the **3C** and MD constructions are shown in Fig 1. The idea of the **3C**-construction has originated from the MAC constructions NMAC and HMAC based on hash functions proposed by Bellare, Canetti and Krawczyk [2]. An important security feature of the NMAC and HMAC functions is to have the *extra application of the compression function and hash function respectively* to prevent extension attacks. The **3C** construction uses the same concept but in a different way. Our construction is named as **3C** as it requires at least 3 executions of the compression function to process arbitrary length messages. That is, to process a single message block using **3C** as a PRF, MAC or as a hash function [1], the compression function is executed three times; first to process the message block, next to process the padded block (Merkle-Damgård strengthening) and finally the XOR accumulation of the intermediate chaining variables as shown in Fig 1.

The salient features of the 3C-construction are listed below:

- **3C** is a single construction, based on the fixed-length-input compression functions, which works as a pseudorandom function (PRF), message authentication code (MAC) and a cryptographic hash function.
- If the compression function is a family of fixed-length-input pseudorandom functions (FI-PRFs) then the **3C**-construction defines a family of variable-length-input pseudorandom functions (VI-PRFs).
- A provably secure **3C**-PRF works as a secure MAC without any additional modifications to the construction.

[1] In the hash function mode, **3C** construction shown in Fig 1 would become **3C-X** a variant of generic **3C** hash function (see Section 5)

– By replacing the key of the **3C**-construction with an initial state (IV) (fixed IV) [2], the **3C**-construction works as a cryptographic hash function.
– The **3C**-MAC function works as an effective variant to the NMAC function. We call this MAC function O-NMAC (one-key NMAC). O-NMAC is a more efficient variant of NMAC and HMAC in the applications where key changes frequently and the key cannot be cached.
– A variant of the O-NMAC function calls the hash function as a black-box serving as a variant for HMAC.
– The features of the **3C**-hash function are given in detail in Section 5 of the paper.

1.1 Motivation and Advantages

The primary motivation behind the proposal **3C** is to show a single construction that works as a PRF, MAC and as a hash function by slightly modifying the Merkle-Damgård iterative structure as shown in Fig 1. Our motivation is to show that *while the consecutive iterations of the compression function gives the speed advantage for a hash function or a MAC function that uses the compression function,* ***the way*** *the compression function is iterated is important for the security of the hash function or a MAC function.* The significant advantages of the **3C** construction from the perspective of PRF, MAC and as a hash function are as follows:

<u>As a PRF:</u>
The **3C**-PRF shows that if the compression function works as a FI-PRF family then one can construct an efficient and secure VI-PRF family. The **3C**-VI-PRF is more efficient than the VI-PRF presented in [3] as their append cascade VI-PRF uses two keys compared to one key of the **3C**-VI-PRF.

<u>As a MAC:</u>
By just assuming that the compression function as a PRF or as a MAC, one cannot guarantee a secure MAC function by iterating it using the Merkle-Damgård iterative principle due to extension attacks. But the **3C**-construction guarantees that by slightly updating the *way* the iterations of the compression function are performed as shown in Fig 1. As said before, *iterating the compression function gives performance advantage whereas the method of iterating the compression function is more important for the security of the final construction.* It is a well known observation that any PRF family is a secure MAC [4,5,12,13]. So, one can use the **3C**-VI-PRF as a secure MAC function and it works as an effective variant to the NMAC function [2]. Unlike NMAC, the **3C**-MAC function uses one key and is just as efficient. A hash variant to the **3C**-MAC function works as a variant to the HMAC function and is also just efficient.

<u>As a Hash Function:</u>
The **3C**-construction shown in Fig 1 works as **3C-X**, a new mode of operation for a hash function. A generic **3C** construction is shown in Section 5 of the paper and **3C-X** works as a simple variant for it. It should be noted that *the current cryptanalytical techniques of converting near-collisions to full collisions in the hash functions MD5 [29], SHA-0 [7] and SHA-1 [28] are insufficient if one instantiates the 3C-construction using the compression functions of these hash functions.* In addition, the **3C-X** construction prevents extension attacks in a more efficient way than the wide-pipe and double-pipe hash functions [26] without increasing the size of the internal state. The **3C** structure offers better protection against multi-block collision attacks than the constructions in [10].

[2] Most hash functions specify the initial state and this initial state is termed as fixed IV [19]

Lucks [26] has proposed double-pipe and wide-pipe hash schemes that improve the resistance against generic attacks in iterated hash functions [14]. One can alter the designs of Lucks in the style of **3C**-hash function improving the security of new constructions not only against multi-block collision attacks but also against multicollision attacks [14] in iterated hash functions. The **3C-X**-hash function is the simplest possible nested mode of operation that one can get by modifying the MD hash function. There has been a lot of work since the attacks on MD5, SHA-0 and SHA-1 in improving the compression functions of these hash functions to resist differential cryptanalysis. See [15, 16, 27]. We note that one can use these strong collision resistant compression functions in the **3C-X**- to get extra protection against differential attacks.

Very recently, Coron *et al.* [8] have provided four hash functions (all are modifications to the plain Merkle-Damgård construction) that work as random oracles when the underlying compression function works as a random oracle. We note that when the underlying compression function works as a random oracle, the **3C**-hash function works as a random oracle. The important point one should note here is while the plain Damgård construction [9] (from which most hash functions evolved starting from MD4) is not a practical construction by itself, the **3C** is a practical construction which preserves the collision resistance of the compression function in a much better way compared to the hash functions that followed the Damgård construction.

1.2 Organization

The paper is organised as follows: In Section 2, notation and definitions are introduced. In Section 3, the proof of security for the **3C**-PRF is given followed by the security analysis of the **3C**-MAC function O-NMAC in Section 4. In Section 5, a generic **3C**-construction as a hash function is introduced and its variant **3C-X** is analysed against multi-block collision attacks on the hash functions. In Section 6, we explained how to combine **3C-X** with the wide-pipe hash function followed by concluding remarks in Section 7.

2 Standard Notation and Definitions

The notations and definitions of this paper shall follow the well established and sensible terminology of [2, 3, 5].

2.1 Pseudorandom functions

Notation: If S is a set (resp. probability space) then $x \stackrel{\$}{\leftarrow} S$ denotes the operation of selecting a message uniformly at random from S (resp. at random according to the underlying distribution of S). If x is represented as blocks $x_1 \ldots x_n$ then $x \stackrel{\$}{\leftarrow} S$ means $x_1 \stackrel{\$}{\leftarrow} S; \ldots ; x_n \stackrel{\$}{\leftarrow} S$. There is always an additional padded data block x_{n+1} appended to the message x containing the binary encoded representation of length of the message x. Let MAPS(X,Y) be the set of all functions mapping from set X to set Y and $|x|$ denote the length of the message string x in bits. A block size of b bits (eg. $b = 512$ bits as for SHA-1) is denoted as $B = \{0, 1\}^b$.
Function Families and their pseudorandomness: A public finite function family F is a finite set of functions and an associated set of keys with a probability distribution on them. All functions in F are assumed to have the same domain and range. Let $\mathrm{Dom}(F)$ be the domain and $\mathrm{Ran}(F)$ be the range of all functions in F.

Let $\{0,1\}^k$ be the set of keys for some integer k called key length. Each key from the set $\{0,1\}^k$ names a function in F and the function corresponding to a key K is denoted as F_K. Picking a function g at random from F (denoted as $g \stackrel{\$}{\leftarrow} F$) means picking a key K uniformly at random from $\{0,1\}^k$ (denoted as $K \stackrel{\$}{\leftarrow} \{0,1\}^k$) and setting $g = F_K$. Given a key, the corresponding function can be computed efficiently on x and is denoted as $F_K(x) = F(K,x)$. Let Time(F) denote the computation time of a program to compute $F_K(x)$ for a given K and x with $|x| \leq n$.

In the keyed function families we consider, Ran(F) is fixed and equals the domain of the keys, $\{0,1\}^k$. The domain Dom(F) can vary. However, the domains that vary over a particular block size b are considered and hence domains will be of form: B (only one block of b bits), B^* (arbitrary finite length strings), B^l(l block strings) and $B^{\leq l}$ (strings of at most l blocks).

A finite function family F is pseudorandom if the input-output behaviour of a random instance $F_K = g$ of F "looks random" to someone who does not know the secret key K. This is formalized via the notion of statistical tests [13] or distinguishers [3]. Formally, for such a test or distinguisher D let

$$\mathrm{Adv}_F^{\mathrm{prf}}(D) = \Pr_{g \stackrel{\$}{\leftarrow} F}[D^g = 1] - \Pr_{g \stackrel{\$}{\leftarrow} \mathrm{MAPS}}[D^g = 1]$$

with the probabilities taken over the choices of g and the coin tosses of D.
The security of family F as a pseudorandom function depends on the resources that D uses that include running-time and number and length of oracle queries. The running time t includes the time taken to execute $g \stackrel{\$}{\leftarrow} F$, time taken to compute responses to oracle queries made by D and the memory which includes the size of the description of D. The distinguisher $D(t,q,l,\epsilon)$ distinguishes F from MAPS(X,Y) if it runs for time t, makes q oracle queries each of length at most l blocks of b bits each and $\mathrm{Adv}_F^{\mathrm{prf}}(D) \leq \epsilon$ where ϵ is called the distinguishing probability. In other words, $D(t,q,l,\epsilon)$-breaks F if $D(t,q,l,\epsilon)$ distinguishes F from MAPS(X,Y). The family F is said to be (t,q,l,ϵ)-secure if there is no distinguisher that (t,q,l,ϵ)-breaks F.

2.2 Message authentication codes

Here we provide formal definition for message authentication codes (MACs) following [5]. We later use these definitions to provide the *concrete security analysis* of the **3C**-MAC function with respect to the compression function.

A MAC is a family of functions defined as $\mathbf{MAC} : \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^\tau$ where $\{0,1\}^*$ is the domain which contains arbitrary strings of finite length that are authenticated using the **MAC** function and $\{0,1\}^\tau$ is the length of the authentication tag Tag. The secret key K of length k is shared between the sender and the receiver. When the sender wants to send the message M to the receiver, it computes the tag Tag of M using the function $\mathbf{MAC}_K(M) = Tag$ and transmits the pair (M, Tag) to the receiver. The receiver verifies the integrity and authenticity of the message M by recomputing authentication tag $\mathbf{MAC}_K(M)$ [3].

The goal of a MAC scheme is to prevent forgery by an adversary that finds a message not seen by the legitimate parties and its corresponding tag. The adversary obtains MACs of messages of its choice by performing a chosen message attack or adaptive chosen message attack. The adversary breaks the **MAC** if it can find a message M which is not part of its queries but from the same

[3] In general, MAC schemes can involve using state information and random nonces instead of just appending the tag. For concreteness we stick to simple stateless or deterministic MACs

domain as queries and its corresponding authentication tag $\mathbf{MAC}_K(M)$. Formally, the success of an adversary A is measured by the following experiment:

Experiment Forge($\mathbf{MAC}$,A)
$K \xleftarrow{\$} \{0,1\}^k$
$(M, Tag) \leftarrow A^{\mathbf{MAC}_K(.)}$
If $\mathbf{MAC}_K(M) = Tag$ and M was not a query of A to its oracle then return 1 else return 0.

In the above experiment, $A^{\mathbf{MAC}_K(.)}$ denotes that A is given oracle for $\mathbf{MAC}_K(.)$ which A can call on any message of its choice. If A is successful then the experiment returns 1 else it retuns 0. The success probability of the adversary is the probability with which it breaks $\mathbf{MAC}$.

The security of a MAC scheme is quantified in terms of the success probability achievable as a function of total number of queries q made in the experiment, running time t of the experiment (which includes the time to choose the key and answer oracle queries, memory and size of the code of the adversary's algorithm) and the maximum length l of each query.

2.3 NMAC and HMAC functions

NMAC and HMAC [2] are the MAC functions proposed by Bellare et. al using cryptographic hash functions. The NMAC function including its security proof was given in [2] and the function is defined as follows:

$$\mathrm{NMAC}_K(x) = f_{K_1}(F_{K_2}(x)). \tag{1}$$

where two random and independent keys K_1 and K_2 each of length k are derived from the key K. The security analysis [2] for the construction shown in (1) adopted the following methodology:
If there exists an algorithm that forges the MAC function NMAC with some significant probability p, then one can explicitly show an algorithm that using the same resources finds collisions for the underlying keyed iterated hash function with at least probability of $p/2$ and forges the keyed compression function as a MAC with at least probability $p/2$.

HMAC is a "fixed IV" variant of NMAC and uses the hash function F as a black box. The HMAC function that works on an arbitrary length message x is defined as:

$$\mathrm{HMAC}_K(x) = F_{IV}(\overline{K} \oplus \mathrm{opad}, F_{IV}(\overline{K} \oplus \mathrm{ipad}, x)) \tag{2}$$

HMAC is a particular case of NMAC and both can be related as $\mathrm{HMAC}_K(x) = \mathrm{NMAC}_{K_1,K_2}(x)$ where $K_1 = f(\overline{K} \oplus \mathrm{opad})$ and $K_2 = f(\overline{K} \oplus \mathrm{ipad})$, f is the compression function of the hash function, opad and ipad are the repetitions of the bytes $0x36$ and $0x5c$ as many times as needed to get a b-bit block and $\overline{K}$ indicates the completion of the key K to a b-bit block by padding K with 0's. The security analysis provided for NMAC applies to HMAC under the assumption that the compression function used to derive the keys K_1 and K_2 for HMAC works as a pseudorandom function [2].

2.4 The basic cascade construction

The basic cascade construction which originates from the collision resistant hash function constructions of Merkle [20] and Damgård [9] is given below.

Let F be the compression function or family of functions mapping from $\{0,1\}^b$ to $\{0,1\}^k$. Here we consider keyed compression functions. A function from the family F is selected by a key K chosen from the set of keys $\{0,1\}^k$. The output length of the compression function is same as the key length k. The families $F^{(1)}, F^{(2)}, F^{(3)}, \ldots$ are defined as follows. The members of $F^{(l)}$ take inputs which are at most l blocks long; that is, the domain of the family $F^{(l)}$ $\mathrm{Dom}(F^{(l)}) = B^{\leq l}$ and $F^{(l)}$ is the l-th iteration of F where $l \in \mathbf{N}$. Let n be the total number of data blocks excluding the last padded block ($n + 1^{\mathrm{th}}$) processed by a function family ; i.e, $n + 1 \leq l$. Then
$F^{(l)}$ is defined as follows assuming that all inputs are non-empty:
$F^{(l)}(K, x_1, x_2, \ldots, x_n) = F(F^{(l)}(K, x_1, x_2, \ldots, x_{n-1}, x_n), x_{n+1})$.
That is for $n \geq 1$ and $n \leq l$, $F^{(l)}(x_1, \ldots, x_n, x_{n+1})$ is computed as follows:

$K_0 \leftarrow K$
for$(i = 1, \ldots, n)$ **do:** $K_i \leftarrow F(K_{i-1}, x_i)$
Output K_n

Note that $F^{(l)}(K, x_1) = F(K, x_1) = F_K(x_1)$.
The iterations of F are denoted with $F^{(*)}$. That is $F^{(*)}(K, x_1, \ldots, x_n, x_{n+1}) \stackrel{\mathrm{def}}{=} F_K^{(n)}(x_1, \ldots, x_n, x_{n+1})$.
Note that the size of the message blocks is at least twice as large as the size of the key. For example, for the compression function of SHA-1, $B = 512$ bits and $k = 160$ bits and for the compression function of SHA-256, $B = 512$ bits and $k = 256$ bits.

3 The 3C-pseudorandom function family

The basic cascade construction $F^{(*)}$ preserves the pseudorandomness of the compression function F. This can be proved using the notion of *prefix-free distinguishers* [3]. The set of queries that a *prefix-free distinguisher* D asks always form a prefix-free set. The following theorem was proved in [3].

Theorem 1 *Let F be a function family with* $\mathrm{Dom}(F) = B$, $\mathrm{Ran}(F) = \{0,1\}^k$ *and key length is k. Suppose F is $(t', q, 1, \epsilon')$-secure and let $l \geq 1$. Then $F^{(*)}$ is (t, q, l, ϵ'')-secure against prefix-free distinguishers, where*

$$t = t' - cq(l + k + b).(\mathrm{Time}(F) + \log q);$$
$$\epsilon'' \leq ql\epsilon'$$

and c is a small specific constant whose value can be determined from the proof [3].

In the basic cascade construction $F^{(*)}$, the prefix-freeness can be ensured by appropriately encoding the queries; for example by prepending the length of the query to each query in an appropriate way. A machine evaluating such a pseudorandom function needs to store the entire input to find the length of the input before processing the input. But this has a serious limitation when it is used as a MAC function since MAC functions should be able to process arbitrary length data with high speed. A solution to this problem was proposed in [3] using an *append cascade*(ACSC) construction which is defined as $F^{\mathrm{acsc}}_{K.d}(x) \stackrel{\mathrm{def}}{=} F_{F_K^{(*)}(x)}(d)$ where d is the extra key material of length δ appended to the message. Proof of security for this scheme was given under the assumption that d is "unpredictable" by the distinguisher.

Now we propose the **3C**-construction which works as an *alternative and improved solution* to the ACSC construction. Later, the **3C**-construction and its variant are used as MAC variants for NMAC and HMAC functions and by replacing the key in the **3C**-construction with a fixed public initial value (IV), it is used as a cryptographic hash function.

If $F^{(*)}$ is the basic cascade construction family based on F taking inputs of length at most l blocks (including the padded block) and is indexed by the k bit key K returning a k bits of output then we define the **3C**-construction based on F as:

$$F_K^{\text{3C}}(x) \stackrel{def}{=} F_{F_K^{(*)}(x)}(\overline{Z}), \tag{3}$$

where $Z = u_1 \oplus u_2 \oplus \ldots \oplus u_n \oplus u_{n+1} = \sum_{i=1}^{n+1} \oplus u_i$. That is, Z is the XOR operation of all of the intermediate chaining variables u_i for $i = 1, \ldots, n+1$. The padding of Z is denoted by $\overline{Z}$. That is, $\overline{Z} = \text{PAD}(\sum_{i=1}^{n+1} \oplus u_i)$ We call this as *chain-XOR block* which is fed as input to the outer function F which is keyed using the *data dependent key* $F_K^{(*)}(x)$ after all the blocks in the message x are processed. In other words, first process the message x in an iterated manner over the function $F_K^{(*)}$ which itself is based on F_K and use this as the *key* to process the *chain-XOR block* $\overline{Z}$ which is fed as input to the external application F. It should be noted that the padding is performed twice for the **3C**-construction; once for the basic cascade construction $F_K^{(*)}$ and next on Z to get the *chain-XOR block*. For a given FI-PRF or compression function family F, we assume that the value Z is random and unpredictable since all of its terms are outputs of FI-PRFs.

The construction is called **3C** as it requires at least three applications of the compression function to process the message and three being the least when there is only one block that needs to be processed. For example, when there is only one message block, the first application of the compression function processes that single block, the second application processes the padded block and the third application processes the block due to the XOR accumulation of the chaining values. Special cases are covered in the remarks.

Informally, in order to break F_K^{3C}, one has to generate queries such that, when translated to F, one query has to be a prefix of another. In order for this to happen, the latter query has to contain Z of any one of the previous queries as a substring. Note that the term "substring" originates from [3]. A substring here means a contiguous sequence of bits of Z that share an edge at the front with the latter query. Without loss of generality we may assume that Z of size k is unpredictable and the probability for this event to happen is exponentially small in k.

The unpredictability property of Z, in turn, depends on the security of the F^{3C} construction and here we provide rigorous security analysis. Our analysis assumes, for simplicity, that the padding of the message is handled appropriately using well known padding technique for hash functions such as SHA-1 [1]. A similar assumption was made in proving the security of NMAC and HMAC [2].

Theorem 2 *Let F be a function family with $\text{Dom}(F) = B = \{0,1\}^b$ used in constructing the family $F^{(*)}$ which is (t', q, l, ϵ'')-secure against prefix-free distinguishers. Then F^{3C} is (t, q, l, ϵ)-secure where $t = t' - cq\ \text{Time}(F)\ \log\ k$ and $\epsilon \leq \epsilon'' + blq2^{-k}$.*

Proof of Theorem 2

Let X and Y be the domain and range of the finite family $F^{(*)}$ constructed from the FI-PRF F. Let $R \stackrel{def}{=} \text{MAPS}(X, Y)$. Note that the domain and range of F^{3C} is also X and Y except that it uses one iteration more than $F^{(*)}$ constructed from F.

We construct an algorithm U which is given a black-box access to a prefix-free distinguisher D_2 with $\epsilon_2 \stackrel{\text{def}}{=} \text{Adv}^{R}_{F^{3\text{C}}}(D_2)$. Now U defines a distinguisher $D_1 \stackrel{\text{def}}{=} U^{D_2}$ with $\epsilon_1 \stackrel{\text{def}}{=} \text{Adv}^{R}_{F^{(*)}}(D_1)$.

A function g is chosen at random either from $F^{(*)}$ or from R. The distinguisher D_1 with oracle access to the function g runs D_2. The distinguisher D_2 makes each query x. The distinguisher D_1 having oracle access to the function g, answers the queries of D_2 by computing Z after all the message blocks are processed including the padded block and then answers the query x of D_2 with $g(x, Z) = F_{F^*_K(x)}(Z)$ where comma indicates the concatenation. The distinguisher D_1 allocates memory dynamically depending on the number of queries to store the Z values for all queries. If any of the queries of D_2 contain any of the stored values of Z as a substring then output bit '1' which means that g is pseudorandom. Else, output whatever D_2 outputs.

For the concrete analysis, let

$$
\begin{aligned}
P_{2,R} &= \Pr_{g \xleftarrow{\$} R}[D_2^g = 1] \\
P_{2,F} &= \Pr_{g \xleftarrow{\$} F^{3\text{C}}}[D_2^g = 1] \\
P_{1,R} &= \Pr_{g \xleftarrow{\$} R}[D_1^g = 1] \\
P_{1,F} &= \Pr_{g \xleftarrow{\$} F^{(*)}}[D_1^g = 1]
\end{aligned}
$$

Now, if g, the oracle of D_1, is taken at random from R, then the answers given by D_1 to D_2's queries are those of a random function. That is, any two different queries are answered by independently chosen values from the range Y. In addition, if $g \xleftarrow{\$} F^{(*)}$ then the answers given by D_1 to D_2's queries are those of a function $g \xleftarrow{\$} F^{3\text{C}}$.

Now we have,

$$P_{2,F} - P_{2,R} = \epsilon_2$$

When $g \xleftarrow{\$} R$, the maximum probability that any one of D_2's queries contain Z as a substring is that of an *unpredictable* value in $\{0,1\}^k$ appearing as a substring in a given set of q strings each of length $b.l$. This probability is less than or equal to $blq2^{-k}$. Thus, we have $P_{1,R} \leq P_{2,R} + blq2^{-k}$. Moreover, whenever D_2 outputs '1', D_1 outputs '1' as well. Thus $P_{1,F} \geq P_{2,F}$. Finally, we get

$$
\begin{aligned}
\epsilon_1 = P_{1,F} - P_{1,R} \geq P_{2,F} - (P_{2,R} + blq2^{-k}) \\
\implies \epsilon_1 \geq \epsilon_2 - blq2^{-k}
\end{aligned}
$$

Replacing ϵ_1 with ϵ'' and ϵ_2 with ϵ, we get $\epsilon \leq \epsilon'' + blq2^{-k}$ □

Now finally, we get the following result by combining the results of Theorems 1 and 2.

Corollary 1. *Let F be a function family with domain* $\text{Dom}(F) = B = \{0,1\}^b$, *range* $\text{Ran}(F) = \{0,1\}^b$ *and key length k. Suppose F is $(t', q, 1, \epsilon')$-secure and let $l \geq 1$. Then $F^{3\text{C}}$ is (t, q, l, ϵ)-secure where*

$$
\begin{aligned}
t &= t' - cq((k + b + l + \log k)\ \text{Time}(F) + (k + l + b)\log q) \\
\epsilon &\leq lq\epsilon' + blq2^{-k}.
\end{aligned}
$$

□

Remarks:

1. From the above proof, the main result is as follows: If F was $(t', q, 1, \epsilon')$-secure then $F^{(*)}$ is (t, q, l, ϵ'')-secure, for t comparable to t' and $\epsilon'' \leq ql\epsilon'$. If F was $(t', q, 1, \epsilon')$-secure then F^{3C} is (t, q, l, ϵ)-secure, for t comparable to t' and $\epsilon \leq ql\epsilon' + blq2^{-k}$. Clearly, the probability of a successful attack against the family F^{3C} is larger than the probability of successful attack against the underlying family F by roughly a factor of ql. A numerical example showing the tightness of the reduction is given below with some reasonable data.
2. NUMERICAL EXAMPLE. Suppose $k = 160$ bits and $b = 2^9$ and we use F^{3C} over a random function family of the same domain. Let $l = 2^{20}$ blocks and $q = 2^{64}$. Then for an ideal FI-PRF F, $\epsilon' \approx q/2^{-k}$ [3,6]. Then there will be a practical method of distinguishing F^{3C} from a random function family using the same resources as follows . $\epsilon' = 1.262 \times 10^{-29}$ and $\epsilon'' = 2.44 \times 10^{-4}$ and $\epsilon \leq 2.44 \times 10^{-4} + 2^9.2^{20}.2^{64}.2^{-160} = 2.44 \times 10^{-4} + 2^{-67} \approx 2.44 \times 10^{-4}$. Consequently, no adversary will be able after having performed the above test, to distinguish F^{3C} from a random function with advantage as large as 2.44×10^{-4}.
3. The ACSC construction [3] uses two independent keys; one of size k acting as IV for the compression function and the other key of small size (not specified in [3]) δ as a data block. It is unsure how close δ to the size k of the key K (which is 160 bits for the compression function of SHA-1). This is not the case with the F^{3C} construction. The second key which is generated *within the construction* due to the *pseudorandomness of the compression functions* is also of size k.
4. In addition, when the ACSC construction [3] is used as a MAC function it is susceptible to the divide and conquer and slice-by-slice trial key recovery attacks [23–25] which apply to the envelope MAC schemes. In fact, having no constraints on the size of the trail key would make these attacks more efficient on the ACSC MAC function if the size of the trial key is very small (as stated in [3]) and does not equal the size k of the key K. Note that this is not the case with the F^{3C} construction when it is used as a MAC function (O-NMAC) due to the extra application of the compression function (see Section 4 for details).
5. The security proof given above considers the message of arbitrary length which implicitly considers short messages. That is, someone may want to process just a 1-bit message which along with padding can fit in a single block. Even in such a case, a separate block is added to pad that 1-bit message (Merkle-Damgård strengthening) where the padded block represents the length of the 1-bit message. So, the total number of compression function computations to process short messages is also three. This may be a slight drawback of this construction from the efficiency point of view when it is used to process very short messages. As far as we know, many real-time applications require processing of at least one block of data.

4 3C PRF as a MAC function

PRFs make secure MACs and the security reduction is standard [4, 5, 12, 13]. Hence, one can simply use F^{3C} construction based on the compression function as a secure MAC function. We call this **3C**-MAC function as "O-NMAC", One-key NMAC. Nevertheless, for completeness, here we show that the reductions from O-NMAC to F^{3C} VI-PRF family and then from O-NMAC to FI-PRF family F are almost tight and security hardly degrades at all. Here we use the definitions of Section 2.

Theorem 3 *Let F^{3C} construction as a MAC called* O-NMAC : $\{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^k$ *be a VI-PRF family and $q, t, l \geq 1$. Then*

$$\mathrm{Adv}^{\mathrm{mac}}_{\text{O-NMAC}}(q,t) \quad \leq \quad \mathrm{Adv}^{\mathrm{prf}}_{3\mathrm{C}}(q,t') + 2^{-k}$$

where $t' = t + O(lqk)$ *and* l *is the maximum number of* b*-bit blocks in a query.*

The proof of Theorem 3 appears in Appendix A.

Now we provide a theorem, using Theorem 2 and Theorem 3, which shows that if F is a FI-PRF family or a compression function then O-NMAC is a secure MAC.

Theorem 4 *Let* $F : \{0,1\}^k \times \{0,1\}^b \to \{0,1\}^k$ *be a FI-PRF family. Then*

$$\mathrm{Adv}^{\mathrm{mac}}_{\text{O-NMAC}}(q,t) \quad \leq \quad \mathrm{Adv}^{\mathrm{prf}}_{\mathrm{F}}(q,t') + (blq+1)2^{-k}$$

where $t' = t + O(lqk) + cq((k+b+l+\log k)\ \mathrm{Time}(F) + (k+l+b)\log q)$

The proof of Theorem 4 appears in Appendix B.
In the following sections, we define O-NMAC and its variant HMAC-1.

4.1 The O-NMAC function

Here we give an algorithm for O-NMAC following the terminology provided for NMAC in Section 2.3.

If K is a random key of length l to the iterated hash function F, then the MAC function O-NMAC on an arbitrary size message x (split into blocks $x_1, \ldots, x_n$ including the padded block x_{n+1}) works as follows:

$$\text{O-NMAC}_K(x) = f_{F_K(x)}(\overline{Z}) \tag{4}$$

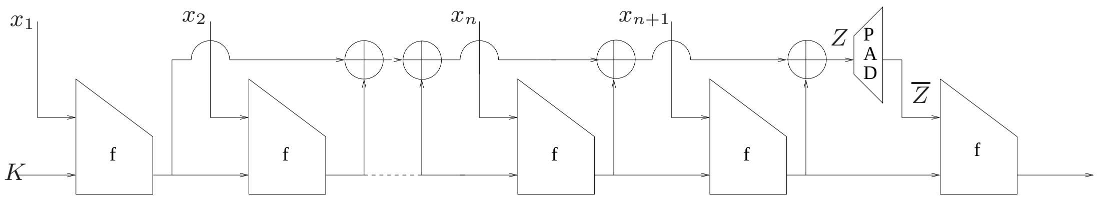

Fig. 2. The O-NMAC construction

The message x to be authenticated using O-NMAC is processed using the hash function F iterated over the compression function f with the secret key K keyed as IV. A standard padding technique (appending x with a bit 1 and some 0's followed by the binary encoded representation of $|x|$) is used for F to pad inputs to an exact multiple of the block length of b bits. In particular, we use the Merkle-Damgård strengthening for the padding of the message x where there is always an extra block x_{n+1} having the binary encoded representation of $|x|$.

In O-NMAC, all the chaining variables are XORed resulting in *chain XOR Z* and Z after padding is fed as an input to the external application f which is *keyed* using the *data dependent key* $F_K(x)$. The *chain XOR Z* is padded using the standard padding technique (appending Z) with a bit 1 and some 0's followed by the binary encoded representation of $|Z|$) to make it a block of b bits. This is denoted with PAD operation in Fig 2. That is, $\mathrm{PAD}(Z) = \overline{Z}$.

4.2 Structural analysis of O-NMAC

In this section, we show the security analysis on the structure of O-NMAC to see whether any cancellation attacks are possible on the structure due to the usage of XOR operation by testing some properties as below. This applies to **3C** PRF as well.

1. **XORing data blocks:** Consider the case of processing one block (or less than one block) messages x_1 and x_2 using O-NMAC with the key K. Then O-NMAC is defined as:
$$\text{O-NMAC}_K(x_1) = f_{F_K(x_1, x_{pad1})}(\overline{Z_1}) \tag{5}$$
$$\text{O-NMAC}_K(x_2) = f_{F_K(x_2, x_{pad2})}(\overline{Z_2}) \tag{6}$$
where $Z_1 = F_K(x_1) \oplus F_{F_K(x_1)}(x_{pad1})$ and $Z_2 = F_K(x_2) \oplus F_{F_K(x_2)}(x_{pad2})$; x_{pad1} and x_{pad2} are padded blocks for messages x_1 and x_2 respectively due to the Merkle-Damgård strengthening. Assume $|x_1| = |x_2|$. Then $x_{pad1} = x_{pad2}$.
Now consider the case of $Z_1 \oplus Z_2$. This implies $F_K(x_1) \oplus F_{F_K(x_1)}(x_{pad1}) \oplus F_K(x_2) \oplus F_{F_K(x_2)}(x_{pad1})$. Clearly, there are no XOR cancellations in this case as for some good compression function, $F_K(x_1) \neq F_K(x_2)$ implying $F_{F_K(x_1)}(x_{pad1}) \neq F_{F_K(x_2)}(x_{pad1})$. This can be generalised to any number of distinct data blocks.
2. **XORing data blocks with first block fixed:** This is a special case of the above case. Consider processing of two block messages x_1, x_2 and x_1, x_2' with their first block x_1 fixed where comma denotes the concatenation. As in the above case, assume $|x_1, x_2| = |x_1, x_2'|$. Then $Z_1 \oplus Z_2$ implies $F_K(x_1) \oplus F_{F_K(x_1)}(x_2) \oplus F_{F_K(x_1,x_2)}(x_{pad1}) \oplus F_K(x_1) \oplus F_{F_K(x_1)}(x_2') \oplus F_{F_K(x_1,x_2')}(x_{pad1})$. This implies $F_{F_K(x_1)}(x_2) \oplus F_{F_K(x_1,x_2)}(x_{pad1}) \oplus F_{F_K(x_1)}(x_2') \oplus F_{F_K(x_1,x_2')}(x_{pad1})$. Clearly this case has now become equivalent to the above case and can be generalised to any number of blocks.

4.3 A fixed IV variant of O-NMAC

Now we introduce a variant of O-NMAC which calls the hash function "as is" like HMAC and we call this variant as HMAC-1. Let F be the iterated and key-less hash function initialized with its usual IV. The function HMAC-1 that works on inputs x of arbitrary length using a single random string K of length k (like for O-NMAC) is defined as follows:

$$\text{HMAC-1}_K(x) = F_{IV}(\overline{F_{IV}(\overline{K}, x)}, \overline{Z}) \tag{7}$$

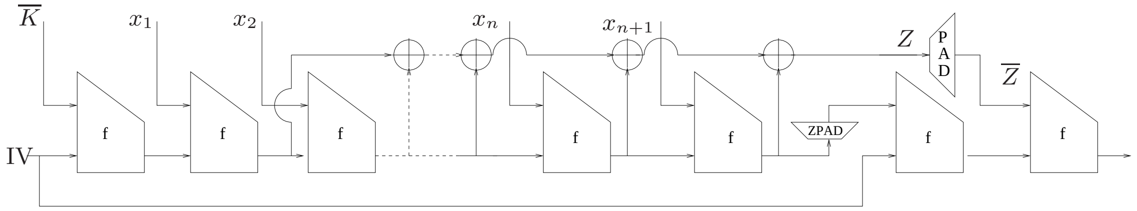

Fig. 3. The HMAC-1 construction

where $\overline{K}$ is the completion of the key K with 0's appended to a full b-bit block-size of the compression function, comma represents concatenation. The PAD operation represents the standard padding operation of the hash function. That is $\mathrm{PAD}(Z) = \overline{Z}$. The operation ZPAD denotes padding with zeros to make the k-bit chaining variable to a b-bit data block. The cost involved in the computation of HMAC and HMAC-1 is the same (see Section 4.4).

4.4 Comparison with NMAC, HMAC and ACSC Constructions

Table 1 shows the comparison of O-NMAC and HMAC-1 with NMAC, HMAC and ACSC [4] constructions in several aspects. The performance of MAC schemes is compared in terms of number of calls to the compression function (CF) of the hash function (HF) to process N blocks of data (there will be an extra block for padding).

Table 1. Comparison among MAC schemes

MAC scheme	# of calls to CF	No. of keys	Security Reduction	Best key-recovery complexity
NMAC	N + 2	2	CF: secure MAC; HF:weakly CRHF	$\approx 2^{k_1} + 2^{k_2}$
HMAC	N + 4	1	CF: secure MAC, PRF; HF:weakly CRHF	2^k
ACSC	N + 1	2	CF: family of FI-PRFs	$\leq 2^k + 2^\delta$
O-NMAC	N + 2	1	CF: family of FI-PRFs	2^k
HMAC-1	N + 4	1	CF: family of FI-PRFs	2^k

Obviously, one main advantage of our MAC functions over NMAC is that they just require management of a single l-bit long key K compared to managing two keys (each of length l) in NMAC that even do not provide security of their combined key length against key-recovery [2] due to the divide and conquer key recovery attack [23–25] on NMAC.

Similarly, when one uses the ACSC construction as a MAC, the sender and receiver of the message have to generate the random secret key along with the shared key K every time they wish to communicate. In O-NMAC, that equivalent secret value Z of size k bits is generated "within" the function and there is no need for the communicating parties to generate and share any other key. Moreover, it was specified in [3] that the size of the second key of the ACSC construction is pretty small but was not clear whether it has the same size as k which, for example, is 160 bits for the keyed compression function of SHA-1.

It should be noted that from the implementation point of view, HMAC requires two extra computations over NMAC which is significant when it is used to authenticate short data streams. It was noted [2] that an implementation can avoid this extra processing by storing the keys K_1 and K_2 (see Section 2.3) as the keys to the NMAC function. This is not the case when the keys vary frequently and an implementation of HMAC cannot avoid this extra over-head by caching the keys when they vary frequently. In such a scenario, the keys K_1 and K_2 need to be generated first and this is accomplished using the compression functions of hash functions such as MD5 and SHA-1 as pseudorandom functions. This would involve two computations of the compression function followed by keying them as IVs for the NMAC function. Here comes the advantage of O-NMAC which just requires a single key. It requires generation of a single key unlike two keys for NMAC.

[4] Note that the intention behind introducing ACSC construction in [3] was to show that the iterations of the compression function preserve pseudorandomness of the compression function but not to show ACSC as a secure MAC. Here we treat it as a secure MAC for comparison

In addition, HMAC-1 does not have to use fixed constants opad and ipad like HMAC. From the perspective of number of calls to the compression function, the cost of HMAC-1 is same as that of HMAC. But in reality, it may be slightly slower than HMAC due to the extra little effort involved in computing Z.

4.5 Applicability of attacks on our MAC functions

One can ask whether the assumption on the compression function can be weakened still producing a secure O-NMAC. Although we cannot answer this question in a formal way, here we provide the security analysis of our constructions in relation to the attacks on MAC schemes based on hash functions independent of the *concrete security analysis* we have provided before. In this analysis, one can assume the compression function of our functions to be a secure MAC or a weakly-collision resistant compression function. These assumptions on the underlying functions were made in the concrete security analysis of NMAC. Note that a secure MAC implies that the MAC function is both collision resistant and one-way for someone who does not know the secret key [22].

- **Extension and key-less collision attacks on hash functions**
  It is well-known that by just assuming that the compression function as a secure MAC one cannot guarantee that the iterated construction $F_k(x)$ is also a secure MAC due to extension attacks [2]. In O-NMAC and HMAC-1, this attack is prevented by the application of the outer compression function.
  An off-line collision attack on the plain hash function leads to collisions on the secret suffix MAC scheme $F_{IV}(x||k)$ where the secret key k is appended to the message [23]. An attacker first finds collisions for two distinct messages x and x' using IV of the hash function such that $F_{IV}(x) = F_{IV}(x')$. Then the attacker requests the oracle of $F_{IV}(x||k)$ the MAC value of x and shows this as the MAC value for the message x'. Note that the attacker cannot find offline collisions in O-NMAC and HMAC-1 as it cannot interact with the legitimate owner of the key. As we will discuss below, though the birthday attacks of finding collisions in hash functions leads to collisions in the iterated MAC schemes, they are not practical for message digest lengths like 224 or 256 bits.
- **Birthday and key recovery attacks**
  The birthday attacks that find collisions in cryptographic hash functions can be applied to MAC schemes based on iterated functions [3, 23]. These attacks apply to NMAC, HMAC, CBC-MAC and our MAC schemes as well. We note that they are the best known forgery attacks on our schemes with the fact that one has to make sure that collisions occur not only in the *cascade chain* but also in the *accumulation chain*. Nevertheless, these attacks are impractical when one uses the compression functions of hash functions such as SHA-256 [1] as it is infeasible to find collisions using birthday attack on these compression functions. This shows that the actual assumptions required by our analysis might be weaker than the pseudorandomness of the compression function. In addition, birthday attacks on keyless hash functions are more efficient than those on MACs based on hash functions as attacks on the latter require interaction with the owner of the key to produce MAC values on a large number of chosen messages with no parallelization.
  These birthday attacks that produce forgery on MAC schemes are extended in the form of "slice by slice trail key recovery" attack [24, 25] to extract the trail key of some versions of the envelope MAC schemes [17, 21]. This attack does not apply to NMAC, HMAC and our MAC

functions as it requires the trail key to be split across the blocks which is not applicable to these constructions. Finally, the divide and conquer key recovery attack [23–25] on the envelope scheme $F(k_1, x, k_2)$ (which applies to NMAC as well [2]) serves to show that the use of a single k-bit key in our constructions does not weaken the functions against exhaustive key search.

5 A new mode of operation for the MD hash function

The generic **3C** hash function construction is shown in Fig 4. This construction has a function f' iterated in the *accumulation chain* and compression function f iterated in the *cascade chain* as in the MD construction. The function f' is a *generic function* which can be instantiated with the compression function f used in the *cascade chain*. We call the generic **3C** hash function with $f' = f$ also as **3C**. The advantage of **3C** over the wide-pipe hash function is the reusage of the compression function used in the *cascade chain* in the *accumulation chain* which makes it easy to deploy. Note that the wide-pipe hash function requires a *complete new compression function design* with an internal state at least twice that of the hash value. Our **3C** hash function also maintains an internal state of size twice that of the size of the hash value in a different way as shown in Fig 4. Implementation of such a construction is trivial for functions with well-structured code like SHA-1 and MD5. In Fig 4, ZPAD denotes the padding of intermediate chaining values with 0's to make them a block size suitable for the compression function in the *accumulation chain*.

The generic **3C**-hash function preserves the collision resistance property of the compression function in a "better way" compared to the hash functions such as SHA-0, SHA-1 and MD5 *depending on the instantiation of f'*. That is, one can use the **3C**-hash function using the compression functions of MD5, SHA-0 and SHA-1 or improved versions to them [15, 16, 27]. We note that *the current techniques of finding near-collisions for the intermediate message blocks is necessary but not sufficient to get a multi-block collision attack on the* ***3C****-hash function*. This is not the case with the currently broken hash functions MD5 and SHA-1 where near-collisions after processing the first message block are converted to full collisions after processing the second message block. The **3C**-hash function uses Merkle-Damgård strengthening to pad the messages. While there has been active research in strengthening the compression functions of widely used hash functions MD5 and SHA-1 [15,16,27], this research focuses on strengthening the Merkle-Damgård construction as practically as possible so that collision resistance of the compression function is extended to its iterations.

The **3C** hash function is less efficient compared to the MD based hash functions though it is as efficient as wide-pipe or double-pipe hash functions. Hence, instead of using the function f for the accumulation process in the *accumulation chain*, one can use a less complex version of the function f (for example half the complexity of f) to make the construction more efficient than the one with $f' = f$. But XOR operation is about as fast as one can get and achieves $n - 1$ resiliency which no non-linear function can do. In addition, instantiating f' with the XOR operation makes the generic **3C** hash function as the simplest efficient modification to the Merkle-Damgård construction. We call this specific case as the **3C-X** hash function.

The following are some of the features of the **3C-X** hash function:

1. The **3C-X**-hash function is the least modification which one can attain by modifying the Merkle-Damgård hash functions without posing much penalty on the performance of the hash function.

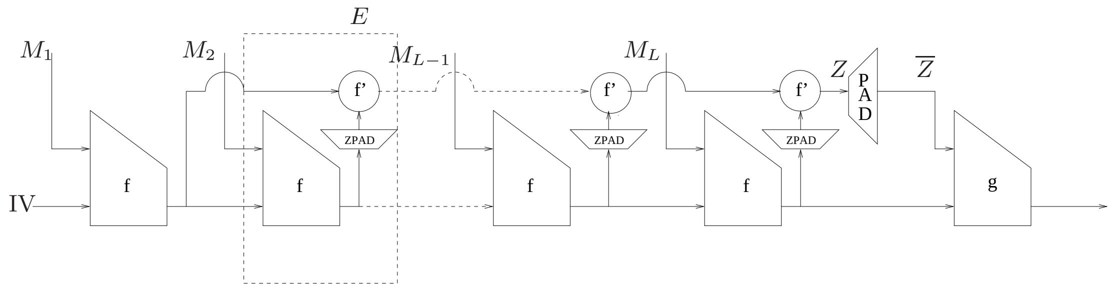

Fig. 4. The generic **3C**-hash function

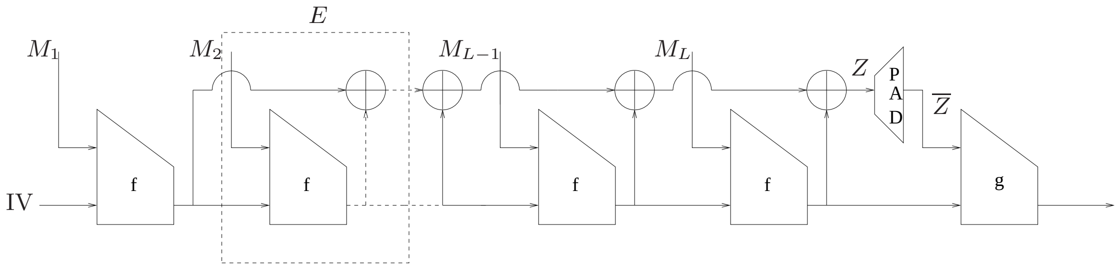

Fig. 5. The **3C-X**-hash function

2. While the Damgård construction [9], for example, is not practical by itself, one of the motivations behind **3C-X** is to design a practical hash function which follows the design motivation of Damgård construction a bit closely than the current dedicated hash functions in the light of recent multi-block collision attacks [7, 28, 29].
3. To process a single message block, the **3C-X**-hash function requires three iterations of the compression function. One for the message block, one for the padded block and the final one for the block due to the XOR accumulation of internal memory as shown in Fig 5.
4. It is possible to use the cryptanalytical techniques used to find collisions in the hash functions MD5 [29], SHA-0 [7] and SHA-1 [28] on the **3C-X**-hash function when it uses the compression function of these hash functions under the condition that collisions should occur at both the chains in the **3C-X**-hash function (Theorem 5). Such a condition is not required to find multi-block collisions on the MD based hash functions.
5. It much more difficult to achieve a simultaneous collision in both the chains of **3C** using the current multi-block collision attack techniques compared to **3C-X** which is a necessary requirement for collisions in these constructions.
6. Extension attacks do not work on the **3C**-hash function due to the application of the additional compression function. Note that **3C** does not perform double hashing unlike the proposals in [10].

In the following section, we provide the security analysis of the generic **3C** hash function followed by the impact of such analysis when one instantiates f' with XOR function and with f in the subsequent remarks.

5.1 Security analysis of 3C hash function

Security analysis for the **3C** hash function can be given in several ways based on the assumptions on the compression function f. By assuming that the function f as a random oracle, one can show that the **3C** works as a random oracle following the assumptions and proof techniques of [8]. In such a case, any application proven secure assuming the hash function as a random oracle would remain secure if one plugs in **3C** assuming that the underlying compression function works as a random oracle. But, assuming the compression function as a random oracle is a stronger assumption than the collision resistance. We will give the security analysis for the **3C** and **3C-X** hash functions from the perspective of recent multi-block collision attacks [7,28,29] followed by the analysis of **3C** against Joux multicollision attacks. It should be noted that these two attacks are different; while a multicollision attack is a generic attack, multi-block collision attack works on the weaknesses in the compression functions of the hash function trying to find collisions for the hash function using near-collisions or pseudo IVs [11]. Multicollisions may involve multi-block collisions anyway. In such a case, the number of possible collisions on the hash function are less compared to the number given by Joux attack [14] as the latter attack requires a collision on every compression function processed. Anyhow, we separate these two attacks in our analysis on our hash functions.

Security analysis of the generic 3C against multi-block collision attacks:
The possible ways of finding collisions to the **3C** hash function F are listed here. We consider two messages $M \neq N$ with lengths L and L' (including padding) respectively such that $F(M) = F(N)$. The messages M and N are expanded to sequences $(M_1, \ldots, M_L) \neq (N_1, \ldots, N_{L'})$ where the last data blocks are the padded blocks containing the lengths of the messages. F_i^M and F_j^N denote the internal hash values obtained while computing $F(M)$ and $F(N)$.

Definition 1. *Consider the* **3C** *hash function F. The terminal and internal collisions on the function F are defined as follows:*

1. *Terminal/Final collisions: They involve one of the following cases:*
   - $F_L^M \neq F_{L'}^N$ *and* $\overline{Z}_L \neq \overline{Z}_{L'}$ *but* $g(F_L^M, \overline{Z}_L) = g(F_{L'}^N, \overline{Z}_{L'})$
   - $F_L^M = F_{L'}^N$ *and* $\overline{Z}_L \neq \overline{Z}_{L'}$ *but* $g(F_L^M, \overline{Z}_L) = g(F_{L'}^N, \overline{Z}_{L'})$
   - $F_L^M \neq F_{L'}^N$ *and* $\overline{Z}_L = \overline{Z}_{L'}$ *but* $g(F_L^M, \overline{Z}_L) = g(F_{L'}^N, \overline{Z}_{L'})$
2. *Internal collisions:* $F_L^M = F_{L'}^N$ *and* $\overline{Z}_L = \overline{Z}_{L'}$, *which implies* $g(F_L^M, \overline{Z_L}) = g(F_L^N, \overline{Z_{L'}})$. □

Definition 2. *Consider the* **3C** *hash function F. An internal collision on the cascade chain implies a collision for f. That is,*
$(F_i^M, M_i) \neq (F_i^N, N_i)$ *with* $f(F_i^M, M_i) = f(F_i^N, N_i)$. □

In this analysis, we treat multi-block collisions on the *cascade chain* as internal collisions on the *cascade chain*. Internal collisions on the *cascade chain* can become terminal collisions if $F_L^M = F_{L'}^N$ and $\overline{Z}_L \neq \overline{Z}_{L'}$ but $g(F_L^M, \overline{Z}_L) = g(F_{L'}^N, \overline{Z}_{L'})$ (see Definition 1). We call the first point (resp. final point) in the *cascade chain* where an internal multi-block collision has occurred as *initial multi-block collision* (resp.*final multi-block collision*).
As pointed out [11], any two pseudo IVs (not necessarily near-collisions) would give a multi-block collision in the Merkle-Damgård hash function. For our analysis of the **3C** hash function we stick to multi-block collisions due to near-collisions. One can give a formal definition for near-collisions in an arbitrary number of ways rather than using hamming distance between collisions [11]. Since no one knows the exact requirement for two chaining values to be nearly collided values, we use the term "related-collisions" instead of "near-collisions".

Definition 3. *If $F(M)$ and $F(N)$ are the digests of two different messages M and N using the hash function F then $F(M)$ and $F(N)$ are related-collisions if $F(M) = T(F(N))$ for some fixed non-trivial function T. The related-collisions come under the category of internal collisions.* $\square$

Theorem 5 *Let F be the* **3C** *hash function designed using a compression function f. Let M and N be two distinct messages. To transform a related collision on the* cascade chain *to a multi-block full collision on the hash function F, it is necessary to have a collision in the* accumulation chain *either at the point of initial multi-block collision or at the point of final multi-block collision in the cascade chain.*

Proof of Theorem 5:
Assume $|M| = |N|$. Let M and N are represented as blocks $M_1, \ldots, M_L$ and $N_1, \ldots, N_L$ where last blocks are the padded blocks. Consider the first related-collision on the t^{th} data block in the *cascade chain* (there can be many related collisions in the *cascade chain* as explained below). That is, $F(M_t) = T(F(N_t))$ (from Definition 3).
Let u_1 and v_1 be the outputs of the first message blocks of the messages M and N and $u_1 \neq v_1$. Let u_i and v_i be the chaining values of the *accumulation chain* for the two messages M and N respectively where $i \in \{2, \ldots, L\}$. Then for $i = 2$ to L, $u_i = u_{i-1} \oplus f(u_{i-1}, M_i)$ and $i = 2$ to L, $v_i = v_{i-1} \oplus f(v_{i-1}, N_i)$.

Assume a terminal collision on F with $u_L \neq v_L$ and $F_L^M = F_L^N$, $g(F_L^M, u_L) = g(F_L^N, v_L)$. Since $F_L^M = F_L^N$, there should have been an internal multi-block collision (since $u_1 \neq v_1$) on the *cascade chain* after processing the t^{th} data block and before the application of the function g due to the related collision on the t^{th} data block in the *cascade chain*.

For a multi-block internal collision on the *cascade chain*, the related collision in the *cascade chain* on the t^{th} data block (it is also possible for some related collisions to take place after the t^{th} data block before a full collision) should have given a full collision after processing any of the blocks $i \in \{t+1, \ldots, L\}$. This also implies that $u_i \neq v_i$ for $i \in \{t, \ldots, L-1\}$ as $u_L \neq v_L$. Without loss of generality, one can also see that $u_i \neq v_i$ for $i \in \{2, \ldots, t-1\}$.

Therefore, for an internal multi-block collision on the *cascade chain* to be transformed to a full collision on the hash function F, it is necessary that the collision to F be a non-terminal collision. That is, u_L must equal v_L. Note that F_L^M equals F_L^N due to an internal multi-block collision on the *cascade chain* which is based on a related-collision on the t^{th} block. For u_L to equal v_L, there should have been a collision at some point on the *accumulation chain*. Assuming that the chaining values of two messages M and N affect the *accumulation chain* at least until the blocks that result in an internal multi-block collision, it is necessary that the *accumulation chains* collide either at the *initial multi-block collision point* or at the *final multi-block collision* point on the *cascade chain* to have a full collision for F.

With the above reasoning, for the condition $F(M) = F(N)$ to be satisfied for the hash function F, having $F(M_t) = T(F(N_t))$ is not sufficient. One must also get a collision on the *accumulation chain* as its accumulation is used as input to the last compression function. So a collision at the same point in both the chains would definitely transform an internal multi-block collision to a full collision on the hash function F. A collision on the *accumulation chain* at the point of final multi-block collision would also transform to a full collision on the hash function F. Anyhow, the later case depends on the messages being chosen by the attacker to carry the initial multi-block collision until a collision has been found on the *accumulation chain*.

Assume an initial internal multi-block collision on the $t+1^{\text{th}}$ block due to $F(M_t) = T(F(N_t)$.
For $F(M_t) = T(F(N_t)) \Rightarrow F(M) = F(N)$ to be satisfied, the following is a necessary and sufficient condition.
$f'(F(M_{t+i}), u_{t+j}) = f'(F(N_{t+i}), v_{t+j})$ where $i, j \in \{1, \ldots, L\}$
Hence to transform a related collision on the *cascade chain* to a multi-block full collision on the hash function F, it is necessary to have a collision in the *accumulation chain* of F either at the point of *initial multi-block collision* or at the point of *final multi-block collision*. □
Remarks:

1. The above analysis assumes $|M| = |N|$. In such a case, $M_L = N_L$. On such equal length messages collisions without the padding clearly lead to collisions with padding. But if $|M| \neq |N|$ then $M_L \neq N_L$. In this case, one has to make sure that the compression functions that process the padded blocks M_L and N_L would still result in a collision for F though there were prior collisions before processing the padded blocks. This applies to Merkle-Damgård construction as well.
2. Note that the analysis described above considers multi-block collisions as internal collisions. Collisions could occur on the final compression function with no internal collisions at all. These collisions come under the category of terminal collisions (see Definition 1).
3. The above analysis assumes terminal collisions due to only one case; $u_L \neq v_L$ and $F_L^M = F_L^N$, $g(F_L^M, u_L) = g(F_L^N, v_L)$. One can also show the analysis by considering the others two cases of Definition 1.
4. The practical implication of the above theorem depends on the strengths of the functions used for f' in the *accumulation chain* and f in the *cascade chain* of the **3C**-hash function.
   - If the function f' is the simple XOR operation, then the necessary and sufficient condition for a collision on **3C-X** from the proof is $F(M_{t+i}) \oplus u_{t+j} = F(N_{t+i}) \oplus v_{t+j}$ where $i, j \in \{1, \ldots, L\}$.
   - Assume a multi-block collision on the *cascade chain* of the **3C-X** hash function with a complexity of 2^t. For example, if the function f is a compression function of SHA-0, then from [7], a 4-block collision can be found on the *cascade chain* with $t = 51$. To transform such a collision attack on the *cascade chain* on to **3C-X**, the total complexity would be $2^t \pm 2^{t'}$ where $2^{t'}$ is the complexity of finding a collision in the *accumulation chain* and $2^{t'} > 0$ or $2^{t'} \leq 0$. Here $2^{t'} = 0$ implies a collision on both the chains of **3C-X** without having to add extra number of blocks after a collision on the *cascade chain*. If $2^{t'} > 0$ then additional number of blocks are appended to the collided blocks on the *cascade chain* to get a multi-block full collision on **3C-X**. This implies that extending the collision resistance of the function f in a slightly better way compared to the MD based hash function such as SHA-0. Note that if the attacker tries to find collisions on **3C-X** aiming $2^{t'} \leq 0$, the attacker is most likely trying to improve the collision search on the *cascade chain* itself which is basically MD construction. For example, assume f in **3C-X** as the compression function of MD5 which is not resistant against 2-block collisions [29]. Now the attacker aiming for a collision on **3C-X-MD5**, either gets a collision on the first block or second block. A first block collision implies a collision to the compression function with the fixed IV. Since the attacker has less control on the *accumulation chain* compared to the messages

that she chooses to get collisions on the *cascade chain*, this case might be difficult compared to the case with $2^{t'} \geq 0$.

– Now consider the **3C** hash function (i.e $f' = f$). This makes achieving a simultaneous collision in both the chains much harder than achieving one in the *cascade chain*. The reason is, the attacker has very less control on the inputs to the function f in the *accumulation chain* than when f' in the *accumulation chain* is a XOR operation. Assume again that the function f in both the chains to be the compression function of SHA-0. To convert the current multi-block collision attack on SHA-0 [7] with a complexity of $2^t = 2^{51}$ on to the **3C** hash function, the attacker has to perform one of the following:
  - Make sure to relate the collisions in both the chains starting from the first compression function to have a simultaneous collision at the end. This would be much harder as the attacker has very less control on the inputs to the function f in the *accumulation chain*. The complexity of such an attack as in the above case, would be $2^t + 2^{t'}$ where the value of t' depends on the cryptanalytical technique used by the attacker and it should be less than $2^{n/2}$, otherwise it would be a birthday attack on the *accumulation chain* (see below).
  - Append extra number of same blocks to the collided chaining values on the *cascade chain* until a collision is found on the *accumulation chain*. This is similar to performing birthday attack on the *accumulation chain* testing every message block on the function f in the *cascade chain*. We conjecture that the total complexity of such an attack would be about $2^t + 2^{n/2}$. If the function f is the compression function of SHA-0, then the total complexity of finding a collision on **3C** using the current techniques [7] is $2^{51} + 2^{80}$ which is much larger than performing birthday attack on **3C** as a black-box with the complexity 2^{80}.

  From this discussion, one can easily see that the best away to find collisions on **3C** is to find collisions on the *cascade chain* for the fixed IV instead of using related-collisions. The hash functions MD5, SHA-0 and SHA-1 are resistant against fixed IV collision attacks but not against pseudo or near-collisions. Hence one can use the fixed-IV collision resistant compressions functions in the **3C** hash function.

5.2 Security analysis of 3C against multicollision attacks

Joux [14] has shown the weakness of Merkle-Damgård construction by constructing multicollisions for the hash functions where D different messages map to a single digest. This generic $D = 2^d$-collision attack requires all intermediate hash values to be equal (of course there are some generalizations [14]). This attack applies to the **3C** hash function as well. This implies that a collision on every compression function in the *cascade chain* would result in a collision at the subsequent point in the *accumulation chain*. From this perspective, if the compression function of **3C** is modeled as a random oracle, as an upper bound on the security of **3C**, the complexity of the total attack is $O(d * 2^{n/2})$ where n is the size of the chaining values and the hash value. But this attack technique is valid on the condition that all the *intermediate hash values* in the *accumulation chain* and in the *cascade chain* are equal. This is valid for any specific case of generic **3C** hash function such as **3C-X** and **3C** itself.

One can combine any instantiation of the generic **3C** hash function with the wide-pipe hash function [26] to improve the resistance against both multi-block collision and multicollision attacks. The **3C-X** hash function combined with the wide-pipe hash function is shown in Section 6.

5.3 Security analysis of 3C against D-way (2^{nd}) preimage attacks

Joux [14] pointed out that one can use multicollision attack technique as a tool to find D-way (2^{nd}) preimages for an MD hash function. For a given hash value $Y \in \{0,1\}^n$, the attacker first finds 2^d collisions on d-block messages $M^1, \ldots, M^{2^d}$ with $H_d = H(M^1) = \ldots = H(M^{2^d})$. Then she finds the block M_{d+1} such that $f(H_d, M_{d+1}) = Y$. The total complexity of the attack is $O(d * 2^{n/2} + 2^n)$. For a 2^{nd} preimage attack with the target message M, just set $Y = H(M)$.

We note that such an attack technique used to find D-way (2^{nd}) preimages for the MD hash function for a given hash value *does* work on any instantiation of the **3C** hash function. For example on **3C-X**, the attacker first finds D collisions on d-block messages $M^1, \ldots, M^{2^d}$ with $H_d = H(M^1) = \ldots = H(M^{2^d})$ with a complexity of $O(d*2^{n/2})$. Then she finds the block M_{d+1} such that the execution of last two compression functions would result in the given digest Y. The later task takes time $O(2^{n+1})$. Hence the total cost to find D preimages for **3C-X** is $O(d * 2^{n/2} + 2^{n+1})$. The same attack on **3C** would take time $O(d*2^{n/2}+3*2^n)$ as for every iteration of f in the *cascade chain* there exists a corresponding iteration of f in the *accumulation chain*. For a D 2^{nd} pre-image attack with the target message M, just set $Y = F(M)$.

5.4 Comparison with other hash function proposals

Ferguson and Schneier proposed a scheme [10] which is basically the NMAC construction with secret keys replaced by the initial states of the hash functions to prevent extension attacks. The scheme is as follows: $F_{IV}(F_{IV}(x))$. It is obvious that Joux multicollision attack [14] applies to their hash function as the application of extra compression function does not prevent extending the multicollisions of the inner function to the total hash function. The complexity of such an attack like on the MD hash is again at most $O(d.2^{n/2})$.

The technique used to find D collisions can be extended to find D-way (2^{nd}) preimages on their scheme. The attack works as follows:

1. The attacker first finds 2^d colliding D-block messages $M_1, \ldots, M_{2^d}$ with $F_d = F(M_1) = \ldots = F(M_{2^d})$. Finding D collisions for the function F_{IV} takes about time $d.2^{n/2}$.
2. For a given hash value Y ($|Y|$=n), the attacker then finds the preimage (the data block) of the external compression function using the IV and the hash value Y in time $O(2^n)$. Let that block be Y'. Assuming that a b-bit compression function is used for the hash function $F_{IV}(F_{IV}(.))$, there would be $|b-n|$ number of padding bits (pad) in Y'. Let $Y' = t||pad$.
3. The attacker then finds a message block M_{d+1}, such that $f(F_d, M_{d+1}) = t$. This takes about the time of $O(2^n)$.

So the total complexity of the attack is $O(d.2^{n/2} + 2^{n+1})$. For a D 2^{nd} pre-image attack with the target message M, just set $Y = F(M)$. Note that this construction involves double hashing (two fixed IVs) whereas **3C** involves single hashing as there is only one instantiation of the hash function with one fixed IV.
Lucks [26] has proposed wide-pipe and double-pipe hash designs as failure-tolerant hash functions showing that these two new designs provide more resistance against generic attacks [14, 18]. Similarly, the **3C**-hash function can be seen as another failure-tolerant hash function. The wide-pipe hash design requires a compression function which is stronger than the hash function itself. This is not the case with the **3C**-hash function. All these functions resist the straight-forward extension attacks as long as their respective design criteria is satisfied; for example, the size of the internal state in the wide-pipe hash should be at least twice that of the size of the hash value.

6 Combining the 3C hash function with other designs

One can convert any instantiation of **3C**-hash function to the style of wide-pipe and double-pipe designs and vice-versa thus enhancing the protection of the hybrid structure against differential cryptanalysis and also against generic attacks.

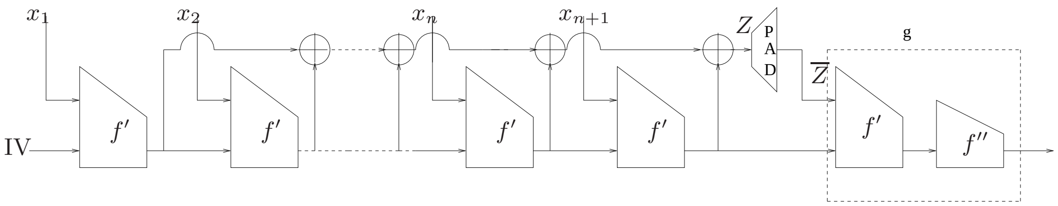

Fig. 6. The **3C-X** wide-pipe hash function

For example, Fig 5 shows the combination of **3C-X** and the wide-pipe design. It uses two compression functions defined as follows:

$f' : \{0,1\}^w \times \{0,1\}^m \rightarrow \{0,1\}^w$ and $f'' : \{0,1\}^w \rightarrow \{0,1\}^n$

$g : \{0,1\}^w \times \{0,1\}^m \rightarrow \{0,1\}^n$

The salient feature of this hybrid construction is that it provides more resistance against multi-collision attacks and multi-block collision attacks. We call this construction as the **3C-X**-wide-pipe hash function. While the security analysis against the multi-block collision attacks follows from the **3C**- hash function, the analysis against multicollision attacks follows from [26].

7 Conclusion

In this paper we proposed a construction called **3C** which works as a PRF, MAC and as a new mode of operation for cryptographic hash function. As a PRF function, our construction offers better security than the append cascade (ACSC) pseudorandom function without having to use two keys. As a MAC function, the **3C** construction (O-NMAC) works as an effective variant to the NMAC function. Significant security advantages of the **3C**-construction are obtained when it is used as a hash function. A generic **3C** hash function was proposed and its variant **3C-X** is dealt in the paper. The **3C** hash function provides more resistance against multi-block collision attacks than the hash functions based on Merkle-Damgård construction and the best way to find collisions on **3C** is to find collisions for the compression function with the fixed IV. When **3C** is combined with the wide-pipe hash function, the net hybrid combination offers more resistance against different forms of attacks on hash functions. While this a draft version, we will update the paper with more features in the next version considering some of the recent attacks on hash functions.

Acknowledgements:

Many thanks to Kapali Viswanathan for his encouragement and valuable insights on this work. Thanks for advising us to look at Bellare, Canetti and Krawczyk's seminal work [2, 3]. Many thanks to Colin Boyd for organizing the Crypto Reading Group at ISI for every two weeks which has been very useful in getting opinions from the fellow researchers on some of the related papers on which this work is based on.

References

1. National Institute of Standards and Technology (NIST) , Computer Systems Laboratory. Secure Hash Standard. Federal Information Processing Standards Publication (FIPS PUB) 180-2, August 2002.
2. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology—CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 18–22 August 1996. Full version of the paper is available at `"http://www-cse.ucsd.edu/users/mihir/papers/hmac.html"`.
3. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Pseudorandom functions revisited: The cascade construction and its concrete security. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, FOCS'96 (Burlington, VT, October 14-16, 1996)*, pages 514–523. IEEE Computer Society, IEEE Computer Society Press, 1996.
4. Mihir Bellare, Joe Kilian, and Phillip Rogaway. The Security of Cipher Block Chaining. In Yvo G. Desmedt, editor, *Advances in Cryptology - Crypto 94*, volume 839 of *Lecture Notes in Computer Science*, pages 341–358. Springer Verlag, 1994.
5. Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer System Sciences*, 61(3):362–399, 2000.
6. Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, 2000.
7. Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and Reduced SHA-1. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 36–57. Springer, 2005.
8. Jean-Sebastien Coron, Yevgeniy Dodis, Cecile Malinaud, and Prashant Puniya. Merkle-damgard revisited: How to construct a Hash Function. In Victor Shoup, editor, *Advances in Cryptology—CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer, 2005, 14–18 August 2005.
9. Ivan Damgard. A design principle for hash functions. In Gilles Brassard, editor, *Advances in Cryptology: CRYPTO 89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer-Verlag, 1989.
10. Niels Ferguson and Bruce Schneier. *Practical Cryptography*, chapter Hash Functions, pages 83–96. John Wiley & Sons, 2003.
11. Praveen Gauravaram, William Millan, and Juanma Gonzalez Nieto. Some thoughts on collision attacks in the hash functions MD5, SHA-0 and SHA-1. Cryptology ePrint Archive, Report 2005/391, 2005. `http://eprint.iacr.org/`.
12. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions (extended abstract). In G. R. Blakley and David Chaum, editors, *Advances in Cryptology: Proceedings of CRYPTO 84*, volume 196 of *Lecture Notes in Computer Science*, pages 276–288. Springer-Verlag, 1985, 19–22 August 1984.
13. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.
14. Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Matt Franklin, editor, *Advances in Cryptology-CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316, Santa Barbara, California, USA, August 15–19 2004. Springer.
15. Charanjit S. Jutla and Anindya C. Patthak. Is SHA-1 conceptually sound? Cryptology ePrint Archive, Report 2005/350, 2005. `http://eprint.iacr.org/`.
16. Charanjit S. Jutla and Anindya C. Patthak. A simple and provably good code for SHA message expansion. Cryptology ePrint Archive, Report 2005/247, 2005. `http://eprint.iacr.org/`.
17. Burt Kaliski and Matt Robshaw. Message authentication with MD5. *CryptoBytes*, 1(1):5–8, Spring 1995.
18. John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than 2n work. In Victor Shoup, editor, *Advances in Cryptology—CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 474–490. Springer, 2005, 14–18 August 2005.
19. Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*, chapter Hash Functions and Data Integrity, pages 321–383. The CRC Press series on discrete mathematics and its applications. CRC Press, 1997.
20. Ralph Merkle. One way hash functions and DES. In Gilles Brassard, editor, *Advances in Cryptology: CRYPTO 89*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer-Verlag, 1989.
21. Philip W. Metzger and William A. Simpson. RFC 1828: IP authentication using keyed MD5, August 1995. Status: PROPOSED STANDARD.

22. Bart Preneel. *Analysis and design of Cryptographic Hash Functions*. PhD thesis, Katholieke Universiteit Leuven, 1993.
23. Bart Preneel and Paul C. van Oorschot. MDx-MAC and building fast MACs from hash functions. In Don Coppersmith, editor, *Advances in Cryptology—CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 1–14. Springer-Verlag, 27–31 August 1995.
24. Bart Preneel and Paul C. van Oorschot. On the security of two MAC algorithms. In Ueli Maurer, editor, *Advances in Cryptology—EUROCRYPT 96*, volume 1070 of *Lecture Notes in Computer Science*, pages 19–32. Springer-Verlag, 12–16 May 1996.
25. Bart Preneel and Paul C. van Oorschot. On the Security of Iterated Message Authentication Codes. *IEEE Transactions on Information Theory*, 45(1):188–199, 1999.
26. Stefan Lucks. A Failure-Friendly Design Principle for Hash Functions. To appear in AsiaCrypt'2005. For an earlier version of this paper see Report 2004/253 at `http://eprint.iacr.org/`.
27. Michael Szydlo and Yiqun Lisa Yin. Collision-Resistant usage of MD5 and SHA-1 via Message Preprocessing. Cryptology ePrint Archive, Report 2005/248, 2005. `http://eprint.iacr.org/`.
28. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In Victor Shoup, editor, *Advances in Cryptology—CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005, 14–18 August 2005.
29. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.

A Proof of Theorem 3

In this proof we use the exact security definition given for MACs in Section 2.2. Let A be any adversary attacking O-NMAC, making q MAC generation queries to the oracle in experiment Forge(O-NMAC,A) with "running time" at most t (for definitions see Section 2.2). Note that the domain of O-NMAC is of arbitrary finite length. Let domain of O-NMAC be at most l blocks with each block containing b bits. Let MAPS(l, k) be a family of all functions.

We design a distinguisher B that distinguishes O-NMAC from MAPS(l, k) as follows:

$$\mathrm{Adv}^{\mathrm{prf}}_{\text{O-NMAC}}(B) \quad \geq \quad \mathrm{Adv}^{\mathrm{mac}}_{\text{O-NMAC}}(A) - 2^{-k} \tag{8}$$

The distinguisher B is given an oracle for a function $g : \{0,1\}^l \to \{0,1\}^k$. It will run in time t' and make at most q queries to its oracle, with the time measured as discussed in Section 2.1.

It was assumed that the oracle in Experiment Forge(**MAC**,A) is invoked at most q times. Note that the integer q includes in its count the message M that A outputs as its forgery. This means that the total number of queries made by A to its oracle is one less than q. Now we give an algorithm for B which will run A, providing it an environment in which A's oracle queries are answered by B.

Distinguisher B
Initialize an empty set S as: $S \leftarrow \emptyset$
Run A
For $(j = 1, \dots q - 1)$
When A sends its oracle some query M_i: reply $g(M_i)$ to A; $S \leftarrow S \cup M_i$
End **For**
A outputs the pair (M, Tag)
if $g(M) = Tag$ and $M \notin S$ return 1
else return 0
Until A stops.

The security analysis is given as follows:

$$\Pr_{g \overset{\$}{\leftarrow} \text{O-NMAC}}[B^g = 1] \quad = \quad \text{Adv}^{\text{mac}}_{\text{O-NMAC}}(A) \tag{9}$$

$$\Pr_{g \overset{\$}{\leftarrow} \text{MAPS}}[B^g = 1] \quad \leq \quad 2^{-k} \tag{10}$$

In (8), the function g is an instance of O-NMAC. The simulation that A gets from B is exactly that of the experiment Forge(O-NMAC,A). B returns 1 only when A makes a successful forgery, thus we have (8).
In (9), A is in an environment where a random function is used to compute MACs. Since g is a random function, $\Pr[g(M) \ = \ Tag]$ is 2^{-k}, thus we have (9).
Finally, (7) is obtained by substituting (8) and (9) into the definition of $\text{Adv}^{\text{prf}}_{\text{O-NMAC}}(B)$.

B Proof of Theorem 4

We prove Theorem 4 first by applying Theorem 3 and then Corollary 1.
From Theorem 3 we have,

$$\text{Adv}^{\text{mac}}_{\text{O-NMAC}}(q,t) \leq \text{Adv}^{\text{prf}}_{\text{hfb}}(q,t') + 2^{-k} \tag{11}$$

From Corollary 1 we have,

$$\text{Adv}^{\text{prf}}_{\text{hfb}}(q,t) \leq \text{Adv}^{\text{prf}}_{\text{F}}(q,t') + blq2^{-k} \tag{12}$$

Combining Theorem 3 and Corollary 1 we get,

$$\text{Adv}^{\text{mac}}_{\text{O-NMAC}}(q,t) \quad \leq \quad \text{Adv}^{\text{prf}}_{\text{F}}(q,t') + (blq+1)2^{-k}$$

where $t' = t + O(lqk) + cq((k+b+l+\text{log}k)\text{Time}(F) + (k+l+b)\text{log}q)$. □