An extended abstract of this paper appears in *Advances in Cryptology – CRYPTO '99*, Lecture Notes in Computer Science Vol. 1666, M. Wiener ed., Springer-Verlag, 1999. This is the full paper.

Constructing VIL-MACs from FIL-MACs: Message authentication under weakened assumptions

Jee Hea An* Mihir Bellare†

June 1999

Abstract

Many practical MACs are designed by iterating applications of some fixed-input-length (FIL) primitive, namely one like a block cipher or compression function that only applies to data of a fixed length. Existing security analyses of these constructions either require a stronger security property from the FIL primitive (eg. pseudorandomness) than the unforgeability required of the final MAC, or, as in the case of HMAC, make security assumptions about the iterated function itself. In this paper we consider the design of iterated MACs under the (minimal) assumption that the given FIL primitive is itself a MAC. We look at three popular transforms, namely CBC, Feistel and the Merkle-Damgård method, and ask for each whether it preserves unforgeability. We show that the answer is no in the first two cases and yes in the third. The last yields an alternative cryptographic hash function based MAC which is secure under weaker assumptions than existing ones.

Keywords: message authentication, unforgeability, weak collision-resistance, proofs of security.

*Dept. of Computer Science, & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-mail: jeehea@cs.ucsd.edu. URL: http://www-cse.ucsd.edu/~jeehea. Supported by a NSF Graduate Fellowship.

†Dept. of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-Mail: mihir@cs.ucsd.edu. URL: http://www-cse.ucsd.edu/users/mihir. Supported in part by NSF CAREER Award CCR-9624439 and a 1996 Packard Foundation Fellowship in Science and Engineering.

Contents

1 Introduction

Directly (from scratch) designed cryptographic primitives (for example block ciphers or compression functions) are typically "fixed input-length" (FIL): they operate on inputs of some small, fixed length. However, usage calls for "variable input-length" (VIL) primitives: ones that can process inputs of longer, and varying lengths. Much cryptographic effort goes into the problem of transforming FIL primitives to VIL primitives. (To mention just two popular examples: the various modes of operation of block ciphers address this problem when the given FIL primitive is a block cipher and the desired VIL primitive a data encryption scheme; and the Merkle-Damgård iteration method [15, 10] addresses this problem when the given FIL primitive is a collision-resistant compression function and the desired VIL primitive is a collision-resistant hash function.) In this paper, we will address this problem for the design of VIL-MACs in the case where the given FIL primitive is itself a MAC, which corresponds to a weak security assumption on the FIL primitive in this context. Let us begin by recalling some background. We then describe more precisely the problem we consider, its motivation and history, and our results.

1.1 Background

MACs. Recall that a message authentication code (MAC) is the most common mechanism for assuring integrity of data communicated between parties who share a secret key k. A MAC is specified by a function g that takes the key k and data x to produce a tag $\tau = g(k, x)$. The sender transmits (x, τ) and the receiver verifies that $g(k, x) = \tau$. The required security property is *unforgeability*, namely that even under a chosen-message attack, it be computationally infeasible for an adversary (not having the secret key k) be able to create a valid pair (x, τ) which is "new" (meaning x has not already been authenticated by the legitimate parties). As the main tool for ensuring data integrity and access control, much effort goes into the design of (secure and efficient) MACs, and many constructions are known. These include block cipher based MACs like the CBC MAC [1] or XOR MACs [8]; hash function based MACs like HMAC [2] or MDx-MAC [19]; and universal hash function based MACs [9, 22]. Many of the existing constructions of MACs fall into the category of FIL to VIL transforms. For example the CBC MAC iterates applications of a block cipher (the underlying FIL primitive), while hash function based MACs iterate (implicitly or explicitly) applications of the underlying compression function.

Assumptions underlying the transforms. Analyses of existing block cipher based MACs make stronger assumptions on the underlying FIL primitive than the unforgeability required of the final VIL-MAC. For example, security analyses of the CBC or XOR MACs provided in [5, 8] model the underlying block cipher as a pseudorandom function, assumed to be "unpredictable" in the sense of [11], a requirement more stringent than unforgeability.

The security analysis of HMAC[1] provided in [2] makes two assumptions: that the (appropriately keyed) compression function is a MAC and also that the iterated compression function is "weakly collision resistant". Thus, the security of HMAC is not shown to follow from an assumption only about the underlying FIL primitive.

Universal hash function based MACs don't usually fall in the FIL to VIL paradigm, but on the subject of assumptions one should note that they require the use of block ciphers modeled as pseudorandom functions to mask the output of the (unconditionally secure) universal hash function, and thereby use assumptions stronger than unforgeability on the underlying primitives.

[1] To be precise, the security analysis we refer to is that of NMAC, of which HMAC is a variant.

1.2 From FIL-MACs to VIL-MACs

THE PROBLEM. We are interested in obtaining VIL-MACs whose security can be shown to follow from (only) the assumption that the underlying FIL primitive is itself a MAC. In other words, we wish to stay within the standard paradigm of transforming a FIL primitive to a VIL-MAC, but we wish the analysis to make a minimal requirement on the security of the given FIL primitive: it need not be unpredictable, but need only be a MAC itself, namely unforgeable. This is, we feel, a natural and basic question, yet one that surprisingly has not been systematically addressed.

BENEFITS OF REDUCED ASSUMPTIONS. It is possible that an attack against the pseudorandomness of a block cipher may be found, yet not one against its unforgeability. A proof of security for a block cipher based MAC that relied on the pseudorandomness assumption is then rendered void. (This does not mean there is an attack on the MAC, but it means the MAC is not backed by a security guarantee in terms of the cipher.) If, however, the proof of security had only made an unforgeability assumption, it would still stand and lend a security guarantee to the MAC construction. Similarly, collision-resistance of a compression function might be found to fail, but the unforgeability of some keyed version of this function may still be intact. (This is true for example for the compression function of MD5.) Thus, if the security analysis of a (keyed) compression-function based MAC relied only on an unforgeability assumption, the security guarantee on the MAC would remain.

Another possibility enabled by this approach would be to design FIL-MACs from scratch. Since the security requirement is weaker than for block ciphers, we might be able to get FIL-MACs that are faster than block ciphers, and thereby speed up message authentication.

1.3 Our results

The benefit (of a VIL-MAC with a security analysis relying only on the assumption that the FIL primitive is a MAC) would be greatest if the construction were an existing, in use one, whose security could now be justified under a weaker assumption. In that case, existing MAC implementations could be left unmodified, but benefit from an improved security guarantee arising from relying only on a weaker assumption. Accordingly, we focus on existing transforms (or slight variants) and ask whether they preserve unforgeability.

CBC MAC. The first and most natural candidate is the CBC MAC. Recall that given a FIL primitive $f\colon \{0,1\}^\kappa \times \{0,1\}^l \to \{0,1\}^l$ its CBC MAC is the transform CBC[f], taking key $k \in \{0,1\}^\kappa$ and input $x = x_1 \ldots x_n \in \{0,1\}^{ln}$ to return y_n, where $y_i = f(k, y_{i-1} \oplus x_i)$ for $1 \le i \le n$, and $y_0 = 0^l$. We already know that if f is a pseudorandom function then CBC[f] is a secure MAC [5], and the question is whether the assumption that f itself is only a MAC is enough to prove that CBC[f] is a secure MAC. We show that it is not. We do this by exhibiting a f that is a secure MAC, but for which there is an attack showing that CBC[f] is not a secure MAC. (This relies of course on the assumption that some secure FIL-MAC exists, since otherwise the question is void.)

MD METHOD. Next we look at Damgård's method [10] for transforming a keyed compression function $f\colon \{0,1\}^\kappa \times \{0,1\}^{\ell+b} \to \{0,1\}^\ell$ into a full-fledged hash function.[2] Actually our method differs slightly in the way it handles input-length variability, which it does by using another key. Our nested, iterated construction, NI[f], takes keys k_1, k_2 and input $x = x_1 \ldots x_n \in \{0,1\}^{nb}$ to return $f(k_2, y_n \| \langle |x| \rangle)$, where $y_i = f(k_1, y_{i-1} \| x_i)$ for $1 \le i \le n$ and $y_0 = 0^\ell$ and $\langle |x| \rangle$ is the length of x written as a binary string of length exactly b bits.

[2] The construction of Damgård is essentially the same as that of Merkle, except that in the latter, the given compression function is keyless, while in the former, it is keyed. Since MACs are keyed, we must use Damgård's setting here.

Although the construction is (essentially) the one used in the collision-resistant hash setting, the analysis needs to be different. This is because of two central differences between MACs and hash functions: MACs rely for their security on a secret key, while hash functions (which, in the Damgård setting, do use a key) make this key public; and the security properties in question are different (unforgeability for MACs, and collision-resistance for hash functions).

We show that if f is a secure MAC then so is NI[f]. The analysis has several steps. As an intermediate step in the analysis we use the notion of weak-collision resistance of [2], and one of our lemmas provides a connection between this and unforgeability.

An appropriately keyed version of the compression function of any existing cryptographic hash function can play the role of f above, as illustrated in Section 4.1. This provides another solution to the problem of using keyed compression functions to design MACs. In comparison with HMAC, the nested, iterated construction has lower throughput because each iteration of the compression function must use a key. Implementation also requires direct access to the compression function, as opposed to being implementable only by calls to the hash function itself. On the other hand, the loss in performance is low, it is still easy to implement, and the supporting security analysis makes weaker assumptions than that of HMAC.

FEISTEL. The Feistel transform is another commonly used method of increasing the amount of data one can process with a given FIL primitive. The basic transform doubles the input length of a given function f. The security of this transform as a function of the number of rounds r has been extensively analyzed for the problem of transforming a pseudorandom function into a pseudorandom permutation: Luby and Rackoff [14] showed that two rounds do not suffice for this purpose, but three do. We ask whether r rounds of Feistel on a MAC f result in a MAC. The answer is easily seen to be no for $r = 2$. But we also show that it remains no for $r = 3$, meaning that the 3-round Feistel transform that turns pseudorandom functions into pseudorandom permutations does not preserve unforgeability. Furthermore, even more rounds do not appear to help in this regard.

1.4 Related work

The FIL to VIL question that we address for MACs is an instance of a classic one, which has been addressed before for many other primitives and has played an important role in the development of the primitives in question. The attraction of the paradigm is clear: It is easier to design and do security analyses for the "smaller", FIL primitives, and then build the VIL primitive on top of them.

The modes of operation of block ciphers were probably the earliest constructions in this area, but an analysis in the light of this paradigm is relatively recent [4]. Perhaps the best known example is the Merkle-Damgård [15, 10] iteration method used in the case of collision-resistant functions. Another early example is (probabilistic) public-key encryption, where Goldwasser and Micali showed that bit-by-bit encryption of a message preserves semantic security [12]. (The FIL primitive here is encryption of a single bit.) Extensive effort has been put into this problem for the case of pseudorandom functions (the problem is to turn a FIL pseudorandom function into a VIL one) with variants of the CBC (MAC) construction [5, 18] and the cascade construction [3] being solutions. Bellare and Rogaway considered the problem and provided solutions for TCR (target-collision-resistant) hashing [6], a notion of hashing due to Naor and Yung [17] which the latter had called universal one-way hashing.

Curiously, the problem of transforming FIL-MACs to VIL-MACs has not been systematically addressed prior to our work. However, some constructions are implicit. Specifically, Merkle's hash tree construction [16] can be analyzed in the case of MACs. Bellare, Goldreich and Goldwasser

use such a design to build incremental MACs [7], and thus a result saying that the tree design transforms FIL-MACs to VIL-MACs seems implicit here.

2 Definitions

FAMILIES OF FUNCTIONS. A *family of functions* is a map $F\colon \mathit{Keys}(F) \times \mathit{Dom}(F) \to \mathit{Rng}(F)$, where $\mathit{Keys}(F)$ is the set of keys of F; $\mathit{Dom}(F)$ is some set of input messages associated to F; and $\mathit{Rng}(F)$ is the set of output strings associated to F. For each key $k \in \mathit{Keys}(F)$ we let $F_k(\cdot) = F(k, \cdot)$. This is a map from $\mathit{Dom}(F)$ to $\mathit{Rng}(F)$. If $\mathit{Keys}(F) = \{0,1\}^\kappa$ for some κ then the latter is the key-length. If $\mathit{Dom}(F) = \{0,1\}^b$ for some b then b is called the input length.

MACs. A MAC is a family of functions F. It is a FIL-MAC (fixed-input-length MAC) if $\mathit{Dom}(F)$ is $\{0,1\}^b$ for some small constant b, and it is a VIL-MAC (variable input length MAC) if $\mathit{Dom}(F)$ contains strings of many different lengths. The security of a MAC is measured via its resistance to existential forgery under chosen-message attack, following [5], which in turn is a concrete security adaptation to the MAC case of the notion of security for digital signatures of [13]. We consider the following experiment $\mathrm{Forge}(A, F)$ where A is an adversary (forger) who has access to an oracle for $F_k(\cdot)$:

Experiment $\mathrm{Forge}(A, F)$
  $k \leftarrow \mathit{Keys}(F)$; $(m, \tau) \leftarrow A^{F_k(\cdot)}$
  If $F_k(m) = \tau$ and m was not an oracle query of A
    then return 1 else return 0

We denote by $\mathbf{Succ}^{\mathrm{mac}}_F(A)$ the probability that the outcome of the experiment $\mathrm{Forge}(A, F)$ is 1. We associate to F its insecurity function, defined for any integers t, q, μ by

$$\mathbf{InSec}^{\mathrm{mac}}_F(t, q, \mu) \stackrel{\mathrm{def}}{=} \max_A \{\, \mathbf{Succ}^{\mathrm{mac}}_F(A) \,\} .$$

Here the maximum is taken over all adversaries A with "running time" t, "number of queries" q, and "total message length" μ. We put the resource names in quotes because they need to be properly defined, and in doing so we adopt some important conventions. Specifically, resources pertain to the experiment $\mathrm{Forge}(A, F)$ rather than the adversary itself. The "running time" of A is defined as the time taken by the experiment $\mathrm{Forge}(A, F)$ (we call this the "actual running time") plus the size of the code implementing algorithm A, all this measured in some fixed RAM model of computation. We stress that the actual running time includes the time of all operations in the experiment $\mathrm{Forge}(A, F)$; specifically it includes the time for key generation, computation of answers to oracle queries, and even the time for the final verification. To measure the cost of oracle queries we let Q_A be the set of all oracle queries made by A, and let $Q = Q_A \cup \{m\}$ be union of this with the message in the forgery. Then the number of queries q is defined as $|Q|$, meaning m is counted (because of the verification query involved). Note also that consideration of these sets means a repeated query is not double-counted. Similarly the total message length is the sum of the lengths of all messages in Q. These conventions will simplify the treatment of concrete security.

The insecurity function is the maximum likelihood of the security of the message authentication scheme F being compromised by an adversary using the indicated resources. We will speak informally of a "secure MAC"; this means a MAC for which the value of the insecurity function is "low" even for "reasonably high" parameter values. When exactly to call a MAC secure is not something we can pin down ubiquitously, because it is so context dependent. So the term secure will be used only in discussion, and results will be stated in terms of the concrete insecurity functions.

3 The CBC MAC does not preserve unforgeability

Let $f\colon \{0,1\}^\kappa \times \{0,1\}^l \to \{0,1\}^l$ be a family of functions. For any fixed integer $n > 0$ we define the associated CBC MAC. It is the family of functions $\mathrm{CBC}[f]\colon \{0,1\}^\kappa \times \{0,1\}^{ln} \to \{0,1\}^l$ defined as follows:

Algorithm $\mathrm{CBC}[f](k, x_1 \ldots x_n)$
  $y_0 \leftarrow 0^l$
  For $i = 1, \ldots, n$ do $y_i \leftarrow f(k, y_{i-1} \oplus x_i)$
  Return y_n

Here $k \in \{0,1\}^\kappa$ is the key, and x_i is the i-th l-bit block of the input message.

We know that if f is a pseudorandom function then $\mathrm{CBC}[f]$ is a secure MAC [5]. Here we show that the weaker requirement that f itself is only a secure MAC does not suffice to guarantee that $\mathrm{CBC}[f]$ is a secure MAC. Thus, the security of the CBC MAC needs relatively strong assumptions on the underlying primitive.

We stress that the number of message blocks n is fixed. If not, splicing attacks are well-known to break the CBC MAC. But length-variability can be dealt with in a variety of ways (cf. [5, 18]), and since the results we show here are negative, they are only strengthened by the restriction to a fixed n.

We prove our claim by presenting an example of a MAC f which is secure, but for which we can present an attack against $\mathrm{CBC}[f]$. We construct f under the assumption that some secure MAC exists, since otherwise there is no issue here at all.

Assume we have a secure MAC $g\colon \{0,1\}^\kappa \times \{0,1\}^{2m} \to \{0,1\}^m$ whose input length is twice its output length. We set $l = 2m$ and transform g into another MAC $f\colon \{0,1\}^\kappa \times \{0,1\}^l \to \{0,1\}^l$. We show that f is a secure MAC but $\mathrm{CBC}[f]$ is not. Below we present f as taking a κ-bit key k and an l-bit input $a = a_1 \| a_2$ which we view as divided into two m-bit halves.

Algorithm $f(k, a_1 a_2)$
  $\sigma \leftarrow g(k, a_1 a_2)$
  Return σa_1

That is, f_k on any input simply returns g_k on the same input, concatenated with the first half of f_k's input. It should be clear intuitively that f is a secure MAC given that g is a secure MAC, because the output of f contains a secure MAC on the input, and the adversary already knows the data a_1 anyway. The following claim relates the securities more precisely.

Claim 3.1 Let g, f be as above. Then $\mathbf{InSec}^{\mathrm{mac}}_f(t, q, \mu) \leq \mathbf{InSec}^{\mathrm{mac}}_g(t, q, \mu)$.

Proof: Let A_f be any forger attacking f, having running time t, number of queries q, and total message length μ. We design a forger A_g attacking g such that

$$\mathbf{Succ}^{\mathrm{mac}}_g(A_g) \;\geq\; \mathbf{Succ}^{\mathrm{mac}}_f(A_f) \tag{1}$$

and furthermore A_g has the same running time, number of queries and total message length as A_f. The claim follows from the definition of the insecurity function. The algorithm A_g uses A_f as a subroutine, itself replying to the oracle queries of A_f to be able to execute the latter. It is presented below.

Algorithm $A_g^{g(k,\cdot)}$
  For $i = 1, \ldots, q$ do
    $A_f \to x_i$
    Break x_i into two equal length parts so that $x_i = x_{i_1} \| x_{i_2}$
    $A_f \leftarrow g(k, x_i) \| x_{i_1}$
  $A_f \to (s, \tau)$
  Break τ into two equal length parts so that $\tau = \tau_1 \| \tau_2$
  Return (s, τ_1)

Here A_g invokes its oracle $g(k,\cdot)$ to implement $f(k,\cdot)$ and thus can reply to the oracle queries of A_f. It is easy to check that A_g will output a successful forgery of g if A_f outputs a successful forgery of f, and both adversaries use the same resources. Equation (1) follows. ∎

We now show that the CBC MAC method is not secure if we use the function f as the underlying base function. The following claim says that there is an attack on CBC$[f]$, which after obtaining the correct tag of only one chosen message, succeeds in forging the tag of a new message. The attack is for the case $n = 2$ of two block messages, so that both the chosen message and the one whose tag is forged have length $2l$.

Claim 3.2 There is a forger F making a single $2l$-bit query to CBC$[f](k,\cdot)$ and achieving

$$\mathbf{Succ}^{\mathrm{mac}}_{\mathrm{CBC}[f]}(F) = 1 \ .$$

Proof: The attacker F is given an oracle for CBC$[f](k,\cdot)$ and works as follows:

Forger $F^{\mathrm{CBC}[f](k,\cdot)}$
  Let a_1, a_2 be distinct m-bit strings and let $x \leftarrow a_1 a_2 0^m 0^m$
  $\sigma_2 \sigma_1 \leftarrow \mathrm{CBC}[f](k, x)$
  $x'_1 \leftarrow a_1 a_2$; $x'_2 \leftarrow a_1 a_2 \oplus \sigma_1 a_1$; $x' \leftarrow x'_1 x'_2$
  Return $(x', \sigma_1 a_1)$

Here F first defined the $2l$ bit message x. It then obtained its l-bit tag from the oracle, and split it into two halves. It then constructed the l-bit blocks x'_1, x'_2 as shown, and concatenated them to get x', which it output along with the claimed tag $\sigma_1 a_1$.

To show that this is a successful attack, we need to check two things. First that the forgery is valid, meaning $\mathrm{CBC}[f](k, x') = \sigma_1 a_1$, and second that the message x' is new, meaning $x' \neq x$.

Let's begin with the second. We note that the last m bits of x' are $a_2 \oplus a_1$. But F chose a_1, a_2 so that $a_2 \neq a_1$ so $a_2 \oplus a_1 \neq 0^m$. But the last m bits of x are zero. So $x' \neq x$.

Now let us verify that $\mathrm{CBC}[f](k, x') = \sigma_1 a_1$. By the definition of f in terms of g, and by the definition of CBC$[f]$, we have

$$\begin{aligned} \mathrm{CBC}[f](k, x) &= f(k, f(k, a_1 a_2) \oplus 0^m 0^m) \\ &= f(k, g(k, a_1 a_2) a_1) \\ &= g(k, g(k, a_1 a_2) a_1) \| g(k, a_1 a_2) \ . \end{aligned}$$

This implies that $\sigma_1 = g(k, a_1a_2)$ and $\sigma_2 = g(k, \sigma_1 a_1)$ in the above code. Using this we see that

$$\begin{aligned} \mathrm{CBC}[f](k, x') &= f(k, f(k, a_1a_2) \oplus (a_1a_2 \oplus \sigma_1 a_1)) \\ &= f(k, \sigma_1 a_1 \oplus (a_1a_2 \oplus \sigma_1 a_1)) \\ &= f(k, a_1a_2) \\ &= \sigma_1 a_1 \end{aligned}$$

as desired. ∎

The construct f above that makes the CBC MAC fail is certainly somewhat contrived; indeed it is set up to make the CBC MAC fail. Accordingly, one reaction to the above is that it does not tell us anything about the security of, say, DES-CBC, because DES does not behave like the function f above. This reaction is not entirely accurate. The question here is whether the assumption that the underlying cipher is a MAC is sufficient to be able to prove that its CBC is also a MAC. The above says that no such proof can exist. So with regard to DES-CBC, we are saying that its security relies on stronger properties of DES than merely being a MAC, for example pseudorandomness.

4 The NI construction preserves unforgeability

Here we define the nested, iterated transform of a FIL-MAC and show that the result is a VIL-MAC.

4.1 The construction

We are given a family of functions $f\colon \{0,1\}^\kappa \times \{0,1\}^{\ell+b} \to \{0,1\}^\ell$ which takes the form of a (keyed) compression function, and we will associate to this the nested iterated (NI) function NI[f]. The construction is specified in two steps; we first define the iteration of f and then show how to get NI[f] from that. See Figure 1 for the pictorial description.

Construction. As the notation indicates, the input to any instance function $f(k, \cdot)$ of the given family has length $\ell + b$ bits. We view such an input as divided into two parts: a *chaining variable* of length ℓ bits and a data block of length b bits. We associate to f its *iteration*, a family IT[f]: $\{0,1\}^\kappa \times \{0,1\}^{\leq L} \to \{0,1\}^{\ell+b}$, where L is to be defined, and for any key k and string x of length at most L we define:

Algorithm IT[f](k, x)
  $y_0 \leftarrow 0^\ell$
  Break x into b-bit blocks, $x = x_1 \ldots x_n$
  For $i = 1, \ldots, n$ do $y_i \leftarrow f(k, y_{i-1} \| x_i)$
  $a \leftarrow y_n \| \langle |x| \rangle$
  Return a

Above if $|x|$ is not a multiple of b, some appropriate padding mechanism is used to extend it. By $\langle |x| \rangle$ we denote a binary representation of $|x|$ as a string of exactly b bits. This representation is possible as long as $|x| < 2^b$, and so we set the maximum message length to $L = 2^b - 1$. This is hardly a restriction in practice given that typical values of b are large.

Now we define the family NI[f]: $\{0,1\}^{2\kappa} \times \{0,1\}^{\leq L} \to \{0,1\}^\ell$. A key for this family is a pair k_1k_2 of κ-bit keys, and for a string x of length at most L we set:

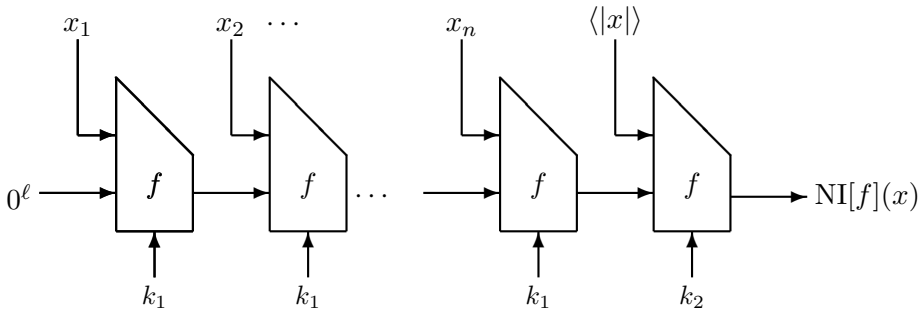

Figure 1: *The nested, iterated construction of a VIL-MAC given a FIL-MAC f.*

Algorithm NI[f](k_1k_2, x)
  $a \leftarrow$ IT[f](k_1, x)
  Return $f(k_2, a)$

Relation to other constructs. Our f has the syntactic form of a (keyed) compression function. The quantity y_n computed in the code of IT[f] is obtained via the iteration method of Damgård [10]; our iterated function is different only in that it appends to this the length of the input x. The main difference is in the security properties. Whereas Damgård assumes f is collision-resistant and wants to show that IT[f] is too, we assume f is a FIL-MAC and want to show NI[f] is a VIL-MAC. The difference is that for MACs the key is secret while for hash functions it is public, and the notions of security are not the same.

Preneel and Van Oorschot [19] suggest that in designing MACs from iterated hash functions, one might use a keyed compression function with a secret key and keyed output transformation. Modulo the handling of length-variability, this is exactly our construction. Preneel et. al. however did not analyze this construction under the assumption that the compression function is a MAC.

Comparing our construction to HMAC/NMAC, the difference, roughly speaking, is that HMAC is based on a hash function (like MD5 or SHA-1) that uses a compression function that is keyless, and iterated in the Merkle style [15]. Had we instead started with a hash function that iterated a keyed compression function in the Damgård style, and applied the HMAC transform to it, we would end up with essentially our construction. This tells us that the Damgård's setting and construction have a nice extra feature not highlighted before: they adapt to the MAC setting in a direct way.

Another difference between our construction and NMAC lies in how the output of the internal functions of the nested functions are formed. Our internal function IT[f] appends the length of the message and the appended length is a part of the function's output whereas F (in NMAC) applies the base function once more on the length of the message.

Instantiation. Appropriately keying the compression function of some existing cryptographic hash function will yield a candidate for f above. For example, let $\mathsf{sha\text{-}1}\colon \{0,1\}^{160+512} \to \{0,1\}^{160}$ be the compression function of SHA-1. We can key it via its 160-bit chaining variable. We would then use the 512 bit regular input as the input of the keyed function. This means we must further subdivide it into two parts, one to play the role of a new chaining variable and another to be the actual data input. This means we set $\kappa = \ell = 160$ and $b = 352$, and define the *keyed sha-1* compression function $\mathsf{ksha\text{-}1}\colon \{0,1\}^{160} \times \{0,1\}^{160+352} \to \{0,1\}^{160}$ by

$$\mathsf{ksha\text{-}1}(k, a\|b) \;=\; \mathsf{sha\text{-}1}(k\|a\|b)\;,$$

for any key $k \in \{0,1\}^{160}$, any $a \in \{0,1\}^{160}$ and any $b \in \{0,1\}^{352}$. Now, we can implement NI[ksha-1] and this will be a secure MAC under the assumption that ksha-1 was a secure MAC on 352 bit

messages.

Note that under this instantiation, each application of sha-1 will process 352 bits of the input, as opposed to 512 in a regular application of sha-1 as used in SHA-1 or HMAC-SHA-1. So the throughput of NI[ksha-1] is a factor of $352/512 \approx 0.69$ times that of HMAC-SHA-1. Also, implementation of NI[ksha-1] calls for access to sha-1; unlike HMAC-SHA-1, it cannot be implemented by calls only to SHA-1. On the other hand, the security of NI[ksha-1] relies on weaker assumptions than that of HMAC-SHA-1. The analysis of the latter assumes that ksha-1 is a secure MAC *and* that the iteration of sha-1 is weakly collision-resistant; the analysis of NI[ksha-1] makes only the former assumption.

4.2 Security analysis

Our assumption is that f above is a secure FIL-MAC. The following theorem says that under this condition (alone) the nested iterated construction based on f is a secure VIL-MAC. The theorem also indicates the concrete security of the transform.

Theorem 4.1 Let $f\colon \{0,1\}^\kappa \times \{0,1\}^{\ell+b} \to \{0,1\}^\ell$ be a fixed input-length MAC. Then the nested, iterated function family $\mathrm{NI}[f]\colon \{0,1\}^{2\kappa} \times \{0,1\}^{\le L} \to \{0,1\}^\ell$ is a variable input-length MAC with

$$\mathbf{InSec}^{\mathrm{mac}}_{\mathrm{NI}[f]}(t,q,\mu) \le \left[1 + \frac{1}{2}\left(\frac{\mu}{b}\right)^2\right] \cdot \mathbf{InSec}^{\mathrm{mac}}_{f}(t',q',\mu')$$

where $t' = t + O(\mu')$, $q' = \mu/b$, and $\mu' = (b+\ell)\cdot\mu/b$.

Tightness of the bound. There is an appreciable loss in security above, with the insecurity of the nested iterated construct being greater than that of the original f by a factor of (roughly) the square of the number μ/b of messages in a chosen-message attack on f. This loss in security is however unavoidable. Iterated constructs of this nature continue to be subject to the birthday attacks illustrated by Preneel and Van Oorschott [19], and these attacks can be used to show that the above bound is essentially tight.

Proof approach. A seemingly natural approach to proving Theorem 4.1 would be to try to imitate the analyses of Merkle and Damgård [15, 10] which showed that transforms very similar to ours preserve collision-resistance. This approach however turned out to be less straightforward to implement here than one might imagine, due to our having to deal with forgeries rather than collisions. Accordingly we take a different approach. We first reduce the question of forgeries to one about a certain kind of collision-resistance, namely "weak-collision resistance", showing that the insecurity of our construct as a MAC can be bounded in terms of its weak collision-resistance and the insecurity of the original f as a MAC. We can bound the weak collision-resistance of the iterated construct in terms of the weak collision-resistance of the original f using the approach of [15, 10], and finally bound the weak-collision resistance of f in terms of its insecurity as a MAC. Putting the three together yields the theorem.

Underlying many of these steps are general lemmas, and we state them in their generality since they might be of independent interest. In particular, we highlight the connections between weak collision-resistance and MACs. We need to begin, however, by saying what is weak collision-resistance.

Weak Collision-resistance. In the usual attack model for finding collisions, the adversary is able to compute the hash function for which it seeks collisions; either it is a single, public function, or, if a family F, the key k (defining the map F_k for which the adversary seeks collisions) is given to the adversary. In the weak collision-resistance setting as defined in [2], the adversary seeking to

find collisions for F_k is not given k, but rather has oracle access to F_k. Weak collision-resistance is thus a less stringent requirement than standard collision-resistance.

Let $F\colon \{0,1\}^\kappa \times \mathit{Dom}(F) \to \mathit{Rng}(F)$ be a family of functions. To formally define its weak collision-resistance we consider the following experiment. Here A is an adversary that gets an oracle for F_k and returns a pair of points m, m' in $\mathit{Dom}(F)$. It wins if these points are a (non-trivial) collision for F_k.

Experiment FindWeakCol(A, F)
$k \leftarrow \mathit{Keys}(F)$; $(m, m') \leftarrow A^{F_k(\cdot)}$
If $m \neq m'$ and $F_k(m) = F_k(m')$ then return 1 else return 0

We denote by $\mathbf{Succ}^{\mathrm{wcr}}_F(A)$ the probability that the above experiment returns 1. We then define

$$\mathbf{InSec}^{\mathrm{wcr}}_F(t, q, \mu) \stackrel{\mathrm{def}}{=} \max_A \{\, \mathbf{Succ}^{\mathrm{wcr}}_F(A) \,\} .$$

As before the maximum is taken over all adversaries A with "running time" t, "number of queries" q, and "total message length" μ, the quantities in quotes being measured with respect to the experiment FindWeakCol(A, F), analogously to the way they were measured in the definition of $\mathbf{InSec}^{\mathrm{mac}}$ described in Section 2. Specifically the running time is the actual execution time of FindWeakCol(A, F) plus the size of the code of A. We let $Q = Q_A \cup \{m, m'\}$ where Q_A is the set of all queries made by A. Then $q = |Q|$ and μ is the sum of the lengths of all messages in Q.

Reduction to WCR. We bound the insecurity of the nested construct as a MAC in terms of its weak-collision resistance and MAC insecurity of the original function. The following generalizes and restates a theorem on NMAC from [2]. In our setting h will be IT$[f]$, and then N becomes NI$[f]$. The proof is an adaptation of the proof in [2], and for completeness is given in Appendix A.1.

Lemma 4.2 Let $f\colon \{0,1\}^\kappa \times \{0,1\}^{\ell+b} \to \{0,1\}^\ell$ be a fixed input-length MAC, and let $h\colon \{0,1\}^\kappa \times D \to \{0,1\}^{\ell+b}$ be a weak collision-resistant function family on some domain D. Define $N\colon \{0,1\}^{2\kappa} \times D \to \{0,1\}^\ell$ via

$$N(k_1 k_2, x) \;=\; f(k_2, h(k_1, x))$$

for any keys $k_1, k_2 \in \{0,1\}^\kappa$ and any $x \in D$. Then N is a MAC with

$$\mathbf{InSec}^{\mathrm{mac}}_N(t, q, \mu) \leq \mathbf{InSec}^{\mathrm{mac}}_f(t, q, q(b+\ell)) \,.+\, \mathbf{InSec}^{\mathrm{wcr}}_h(t, q, \mu)$$

To prove Theorem 4.1 we will apply the above lemma with $h = \mathrm{IT}[f]$. Accordingly our task now reduces to bounding the weak collision-resistance insecurity of IT$[f]$. But remember that we want this bound to be in terms of the insecurity of f as a MAC. We thus obtain the bound in two steps. We first bound the weak collision-resistance of IT$[f]$ in terms of the weak collision-resistance of f, and then bound the latter via its insecurity as a MAC.

Weak Collision-Resistance of IT$[f]$. We now show that if f is a weak collision-resistant function family, then the iterated construction IT$[f]$ is also a weak collision-resistant function family.

Lemma 4.3 Let $f\colon \{0,1\}^\kappa \times \{0,1\}^{\ell+b} \to \{0,1\}^\ell$ be a weak collision-resistant function family. Then,

$$\mathbf{InSec}^{\mathrm{wcr}}_{\mathrm{IT}[f]}(t, q, \mu) \leq \mathbf{InSec}^{\mathrm{wcr}}_f(t, \frac{\mu}{b}, (b+\ell)\frac{\mu}{b})$$

The proof is analogous to those in [15, 10] which analyze similar constructs with regard to standard (not weak) collision-resistance. To extend them one must first observe that their reductions make

only black-box use of the underlying function instances and can thus be implemented in the weak collision-resistance setting via the oracle for the function instance. Second, our way of handling the length variability, although different, can be shown to work. Finally, we provide the concrete security analysis needed to establish the quantitative security claims above. The proof is given in Appendix A.2.

Given the above two lemmas our task has reduced to bounding the weak collision-resistance insecurity of f in terms of its MAC insecurity. The connection is actually much more general.

Weak Collision-Resistance of any MAC. We show that any secure MAC is weakly collision-resistant, although there is a loss in security in relating the two properties. This is actually the main lemma in our proof, and may be of general interest.

Lemma 4.4 Let $g\colon \{0,1\}^\kappa \times \mathit{Dom}(g) \to \{0,1\}^\ell$ be a family of functions. Then,

$$\mathbf{InSec}^{\mathrm{wcr}}_g(t,q,\mu) \leq \frac{q(q-1)}{2} \cdot \mathbf{InSec}^{\mathrm{mac}}_g(t+O(\mu),q,\mu)$$

The proof of the above is given in Appendix A.3.

Proof of Theorem 4.1. We now use the three lemmas above to complete the proof of Theorem 4.1. Letting $\epsilon = \mathbf{InSec}^{\mathrm{mac}}_f(t,q,q(b+\ell))$ for conciseness, we have:

$$\begin{aligned}
\mathbf{InSec}^{\mathrm{mac}}_{\mathrm{NI}[f]}(t,q,\mu) &\leq \epsilon + \mathbf{InSec}^{\mathrm{wcr}}_{\mathrm{IT}[f]}(t,q,\mu) && (2)\\
&\leq \epsilon + \mathbf{InSec}^{\mathrm{wcr}}_f(t,\frac{\mu}{b},(b+\ell)\frac{\mu}{b}) && (3)\\
&\leq \epsilon + \frac{1}{2}\cdot\left(\frac{\mu}{b}\right)^2 \cdot \mathbf{InSec}^{\mathrm{mac}}_f(t',q',\mu') && (4)\\
&\leq \left[1+\frac{1}{2}\left(\frac{\mu}{b}\right)^2\right] \cdot \mathbf{InSec}^{\mathrm{mac}}_f(t',q',\mu')\ , && (5)
\end{aligned}$$

where $t' = t + O(\mu')$, $q' = \mu/b$ and $\mu' = (b+\ell)\cdot\mu/b$. In Equation (2) we used Lemma 4.2. In Equation (3) we used Lemma 4.3. In Equation (4), Lemma 4.4 is used, and the two terms of $\mathbf{InSec}^{\mathrm{mac}}_f$ are added in Equation (5) with the larger of the two resource parameters taken as the final resource parameters to obtain the conclusion of the theorem.

Tightness of Lemma 4.4. It is natural to ask whether the loss in security in Lemma 4.4 is inherent or due to some weakness in the analysis. (This question is particularly relevant since the loss in security in Theorem 4.1 comes entirely from that in Lemma 4.4.) It turns out that the analysis of Lemma 4.4 is the best possible up to a small constant factor. To prove this we give an example of a family of functions $g\colon \{0,1\}^\kappa \times \mathit{Dom}(g) \to \{0,1\}^\ell$ for which

$$\begin{aligned}
\mathbf{InSec}^{\mathrm{wcr}}_g(t,q,\mu) &\geq 0.63 \cdot \frac{q(q-1)}{2}\cdot\frac{1}{M} && (6)\\
\mathbf{InSec}^{\mathrm{mac}}_g(t',q,\mu) &\leq \frac{1}{M}\ , && (7)
\end{aligned}$$

for some value $M > 0$. The two inequalities above indicate that there can be a gap of up to $\Omega(q^2)$ in the insecurities of g viewed as a weakly collision-resistant family and as a MAC. Namely g could be quite secure as a MAC, yet less secure by a factor of about q^2 as a weak collision-resistant family.

The example is simply the family of random functions of D to $\{0,1\}^m$ where D could be any set of size at least q, and m is an arbitrary positive integer. More precisely, $\kappa = m \cdot 2^{|D|}$ and each κ-bit

key k specifies a function of $D \to \{0,1\}^m$ by simply listing the sequence of values this function takes on the points in its domain D, so that $g\colon\ \{0,1\}^\kappa \times D \to \{0,1\}^m$. Notice that the values of t, t' are irrelevant since the success of an adversary in this information-theoretic setting depends only on the number of queries, not on the computation time. To justify the claims, first consider a collision-finder C. In q distinct queries it can find a collision with the probability of the birthday paradox, which is lower bounded by the quantity of Equation (6). (The constant is $1 - 1/e$.) On the other hand, a forger must output a pair (x, τ) where x is unqueried, and if so τ has chance at most $1/M$ of being correct. This gives us Equation (7).

5 Feistel does not preserve unforgeability

Let $f\colon\ \{0,1\}^\kappa \times \{0,1\}^l \to \{0,1\}^l$ be a family of functions. For any fixed integer $r > 0$, we define the r-round Feistel transform. It is a family of functions $\mathrm{FST}^r[f]\colon\ \{0,1\}^{r\kappa} \times \{0,1\}^{2l} \to \{0,1\}^{2l}$. Given keys $k_1, \ldots, k_r$ and input LR where $|L| = |R| = l$, we define

Algorithm $\mathrm{FST}^r[f](k_1 \ldots k_r, LR)$
$L_0 \leftarrow L\ ;\ R_0 \leftarrow R$
For $i = 1, \ldots, r$ do $Z_{i-1} \leftarrow f_{k_i}(R_{i-1})\ ;\ R_i \leftarrow L_{i-1} \oplus Z_{i-1}\ ;\ L_i \leftarrow R_{i-1}$
Return $L_r R_r$

Here $L_i R_i$ is the $2l$-bit block at the end of the i-th round. The Feistel transform has been used extensively to extend (double) the input size of a given pseudorandom function. Luby and Rackoff have shown that $\mathrm{FST}^3[f]$ is a pseudorandom permutation if f is a pseudorandom function [14]. Here we examine the possibility of $\mathrm{FST}^r[f]$ being a secure MAC under the assumption that f is only a secure MAC.

Luby and Rackoff showed that $\mathrm{FST}^2[f]$ is not pseudorandom even if f is pseudorandom [14]. This does not directly tell us anything about whether $\mathrm{FST}^2[f]$ is a secure MAC given that f is a secure MAC. But in fact it is easy to design an attack showing that $\mathrm{FST}^2[f]$ is not a secure MAC even if f is a secure MAC. Note that the following claim is very strong in the sense that it holds for all f: No matter what function family f you start with (in particular, it could be a secure MAC or even pseudorandom), applying the two-round Feistel transform to it results in a family for which there exists a simple attack to break it as a MAC.

Claim 5.1 For any $f\colon\ \{0,1\}^\kappa \times \{0,1\}^l \to \{0,1\}^l$ there is a forger F making two $2l$-bit queries to $\mathrm{FST}^2[f](k_1 k_2, \cdot)$ and achieving

$$\mathbf{Succ}^{\mathrm{mac}}_{\mathrm{FST}^2[f]}(F)\ \geq\ 1 - 2^{-l}\ .$$

Proof: The forger F is given an oracle for $\mathrm{FST}^2[f](k, \cdot)$, where $k = k_1 k_2 \in \{0,1\}^{2\kappa}$ is the key. It proceeds as follows.

Forger $F^{\mathrm{FST}^2[f](k_1 k_2, \cdot)}$
Let L_0^1 be an l-bit string; let $L_0^2 \xleftarrow{R} \{0,1\}^l$; let R_0^1, R_0^2 be any two distinct l-bit strings
$L_2^1 R_2^1 \leftarrow \mathrm{FST}^2[f](k_1 k_2, L_0^1 R_0^1)$
$L_2^2 R_2^2 \leftarrow \mathrm{FST}^2[f](k_1 k_2, L_0^2 R_0^2)$
$Z_0^1 \leftarrow L_0^1 \oplus L_2^1\ ;\ Z_0^2 \leftarrow L_0^2 \oplus L_2^2\ ;\ Z_1^1 \leftarrow R_2^1 \oplus R_0^1$
$L_0 \leftarrow L_0^1 \oplus Z_0^1 \oplus Z_0^2\ ;\ R_0 \leftarrow R_0^2$

$L_2 \leftarrow L_0^1 \oplus Z_0^1$; $R_2 \leftarrow Z_1^1 \oplus R_0^2$
Return (L_0R_0, L_2R_2)

We claim two things. First that $L_0R_0 \notin \{L_0^1R_0^1, L_0^2R_0^2\}$ with probability at least $1 - 2^{-l}$, the probability being over the random choice of L_0^2 made by the forger F. Second that the forgery is valid, meaning $\mathrm{FST}^2[f](k_1k_2, L_0R_0) = L_2R_2$.

For the first claim note that $L_0 = L_0^1 \oplus Z_0^1 \oplus Z_0^2$ does not depend on L_0^2, since $Z_0^1 = f(k_1, R_0^1)$ and $Z_0^2 = f(k_1, R_0^2)$. Thus the probability that $L_0 = L_0^2$, taken over the random choice of L_0^2, is at most 2^{-l}. Thus the probability that $L_0R_0 = L_0^2R_0^2$ is also at most 2^{-l}. On the other hand $R_0 = R_0^2$ and this is different from R_0^1 by the choice of R_0^1, R_0^2 as distinct strings. So $L_0R_0 \neq L_0^1R_0^1$ for sure.

For the second claim, we apply the 2-round Feistel transform to L_0R_0, and simplify using the known quantities related to the transform on our two other points:

$$\begin{aligned} L_0\|R_0 &= L_0^1 \oplus Z_0^1 \oplus Z_0^2 \| R_0^2 \\ Z_0 &= Z_0^2 \\ L_1\|R_1 &= R_0^2 \| L_0^1 \oplus Z_0^1 \\ Z_1 &= Z_1^1 \\ L_2\|R_2 &= L_0^1 \oplus Z_0^1 \| Z_1^1 \oplus R_0^2 \end{aligned}$$

The following table may help to better visualize this. The first row is the input to the 2-round Feistel transform. The second row depicts the result of computing $f(k_1, \cdot)$ on the right hand side of the element in the row above. And so on.

query 1	*query 2*	*forgery*
$L_0^1 \mid R_0^1$	$L_0^2 \mid R_0^2$	$L_0^1 \oplus Z_0^1 \oplus Z_0^2 \mid R_0^2$
$\mid Z_0^1$	$\mid Z_0^2$	$\mid Z_0^2$
$R_0^1 \mid L_0^1 \oplus Z_0^1$	$R_0^2 \mid L_0^2 \oplus Z_0^2$	$R_0^2 \mid L_0^1 \oplus Z_0^1$
$\mid Z_1^1$	$\mid Z_1^2$	$\mid Z_1^1$
$L_0^1 \oplus Z_0^1 \mid R_0^1 \oplus Z_1^1$	$L_0^2 \oplus Z_0^2 \mid R_0^2 \oplus Z_1^2$	$L_0^1 \oplus Z_0^1 \mid R_0^2 \oplus Z_1^1$

This ends the proof. ∎

We now go on to the more interesting case of three rounds, where the transform is known to be pseudorandomness preserving. We show that it is nonetheless not unforgeability preserving in general. Namely the assumption that f is a secure MAC does not suffice to guarantee that $\mathrm{FST}^3[f]$ is a secure MAC. We prove our claim by presenting an attack against $\mathrm{FST}^3[f]$ when f is the MAC of Section 3 for which we had presented an attack against $\mathrm{CBC}[f]$. (Thus, the claim is not as strong as for the two-round case where we were able to show that the two-round Feistel transform of any function is an insecure MAC. Rather, as in Section 3, what we are saying here is that the assumption that f is a secure MAC is provably not enough to show that $\mathrm{FST}^3[f]$ is a secure MAC, because there is an example where f is a secure MAC but $\mathrm{FST}^3[f]$ is not.) Recall that $f\colon \{0,1\}^\kappa \times \{0,1\}^l \to \{0,1\}^l$ was designed in terms of an underlying secure but arbitrary MAC $g\colon \{0,1\}^{2m} \to \{0,1\}^m$, and we set $l = 2m$. Let us now see what happens when we evaluate $\mathrm{FST}^3[f](k_1k_2k_3, L_0R_0)$. We write

$L_0 = a_0\|a_1$ and $R_0 = b_0\|b_1$ where $|a_0| = |a_1| = |b_0| = |b_1| = m = l/2$ bits, and work through the three Feistel rounds, writing the intermediate results in terms of the notation used in describing the Feistel algorithm above:

$$\begin{array}{rcrcl}
L_0 \mid R_0 & = & & a_0\|a_1 \mid b_0\|b_1 \\
\mid Z_0 & = & & \mid \mu\|b_0 \\
L_1 \mid R_1 & = & & b_0\|b_1 \mid a_0 \oplus \mu\|a_1 \oplus b_0 \\
\mid Z_1 & = & & \mid \mu'\|a_0 \oplus \mu \\
L_2 \mid R_2 & = & a_0 \oplus \mu\|a_1 \oplus b_0 & \mid & b_0 \oplus \mu'\|b_1 \oplus a_0 \oplus \mu \\
\mid Z_2 & = & & \mid & \mu''\|b_0 \oplus \mu' \\
L_3 \mid R_3 & = & b_0 \oplus \mu'\|b_1 \oplus a_0 \oplus \mu & \mid & a_0 \oplus \mu \oplus \mu''\|a_1 \oplus \mu'
\end{array}$$

Here we have set

$$\begin{aligned}
\mu &= g(k_1, b_0 b_1) \\
\mu' &= g(k_2, a_0 \oplus \mu\|a_1 \oplus b_0) \\
\mu'' &= g(k_3, b_0 \oplus \mu'\|b_1 \oplus a_0 \oplus \mu)\ .
\end{aligned}$$

Write $L_3 = a_3\|a_3'$ and $R_3 = b_3\|b_3'$. We notice that given the output L_3R_3 and the input L_0R_0, it is possible to extract the values μ, μ', μ'', even without knowledge of any of the keys. Namely $\mu = b_1 \oplus a_0 \oplus a_3'$ and $\mu' = a_3 \oplus b_0$ and $\mu'' = a_0 \oplus \mu \oplus b_3$. Furthermore notice that once an attacker has these values, it can also compute Z_0, Z_1, Z_2, the internal Feistel values. Based on this we will present an attack against $\mathrm{FST}^3[f]$.

Claim 5.2 There is a forger A making four $2l$-bit queries to $\mathrm{FST}^3[f](k,\cdot)$ and achieving

$$\mathbf{Succ}^{\mathrm{mac}}_{\mathrm{FST}^3[f]}(A) \ \geq\ 1 - 4 \cdot 2^{-l}\ .$$

Proof: The attacker A is given an oracle for $\mathrm{FST}^3[f](k,\cdot)$, where $k = k_1k_2k_3 \in \{0,1\}^{3\kappa}$ is the key. It makes the four queries displayed, respectively, as the first rows of the first four columns in Figure 2. The first two queries generated by A are random. A then generates the next two queries adaptively, using the results of the previous queries. Notice that the third and fourth queries are functions of Z-values occurring in the 3-round Feistel computation on the first two queries. The attacker A can obtain these values using the observation above. Finally, A comes up with the forgery (x, τ), where x and τ are displayed, respectively, as the first and last rows in the fifth column of the same Figure.

Notice that in queries 3 and 4 in Figure 2, the rows after certain values (Z_0^3,Z_2^3) are empty. They are omitted for simplicity because only those two values (Z_0^3,Z_2^3) are needed to form the next queries or the forgery, and the rest of the values are not needed. In the actual attack, those values are computed from the outputs of the oracle.

To show that this is a successful attack, we need to check two things. First that the forgery is valid, meaning $\mathrm{FST}^3[f](k,x) = \tau$, and second that the message x is new, meaning $x \notin \{x_1 \ldots x_4\}$.

We can easily see that the forgery is valid by examining the values in the table. The second requirement that x is new can be achieved with high probability if the adversary chooses the strings L_0^1, L_0^2, and L_0^3 randomly. If the said strings are chosen randomly, then the l-bit left-half of

query 1	*query 2*	*query 3*	*query 4*	*forgery*
$L_0^1 \mid R_0^1$	$L_0^2 \mid R_0^2$	$L_0^3 \mid R_0^2 \oplus Z_1^1 \oplus Z_1^2$	$R_1^2 \oplus Z_0^3 \mid R_0^2 \oplus Z_1^1 \oplus Z_1^2$	$R_1^1 \oplus Z_0^2 \mid R_0^2$
$\mid Z_0^1$	$\mid Z_0^2$	$\mid Z_0^3$	$\mid Z_0^3$	$\mid Z_0^2$
$R_0^1 \mid R_1^1$	$R_0^2 \mid R_1^2$	$\mid$	$R_0^2 \oplus Z_1^1 \oplus Z_1^2 \mid R_1^2$	$R_0^2 \mid R_1^1$
$\mid Z_1^1$	$\mid Z_1^2$	$\mid$	$\mid Z_1^2$	$\mid Z_1^1$
$R_1^1 \mid R_2^1$	$R_1^2 \mid R_2^2$	$\mid$	$R_1^2 \mid R_0^2 \oplus Z_1^1$	$R_1^1 \mid R_0^2 \oplus Z_1^1$
$\mid Z_2^1$	$\mid Z_2^2$	$\mid$	$\mid Z_2^3$	$\mid Z_2^3$
$R_2^1 \mid R_3^1$	$R_2^2 \mid R_3^2$	$\mid$	$\mid$	$R_0^2 \oplus Z_1^1 \mid R_1^1 \oplus Z_2^3$

Figure 2: Contents of the queries and the intermediate/final results and the forgery.

each queried string becomes random and the probability of the forgery string x matching any one of the four queried strings is very small, specifically at most $4 \cdot 2^{-l}$. This means that the probability of the forgery being new (and valid) is $1 - 4 \cdot 2^{-l}$ as claimed. ∎

Acknowledgments

Thanks to the Crypto 99 program committee for their comments.

References

[1] ANSI X9.9, "American National Standard for Financial Institution Message Authentication (Wholesale)," American Bankers Association, 1981. Revised 1986.

[2] M. Bellare, R. Canetti and H. Krawczyk, "Keying hash functions for message authentication," *Advances in Cryptology – CRYPTO '96*, Lecture Notes in Computer Science Vol. 1109, N. Koblitz ed., Springer-Verlag, 1996.

[3] M. Bellare, R. Canetti and H. Krawczyk, "Pseudorandom functions revisited: the cascade construction and its concrete security," *Proceedings of the 37th Symposium on Foundations of Computer Science*, IEEE, 1996.

[4] M. Bellare, A. Desai, E. Jokipii and P. Rogaway, "A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation," *Proceedings of the 38th Symposium on Foundations of Computer Science*, IEEE, 1997.

[5] M. Bellare, J. Kilian and P. Rogaway, "The security of cipher block chaining," *Advances in Cryptology – CRYPTO '94*, Lecture Notes in Computer Science Vol. 839, Y. Desmedt ed., Springer-Verlag, 1994.

[6] M. Bellare and P. Rogaway, "Collision-Resistant Hashing: Towards Making UOWHFs Practical," *Advances in Cryptology – CRYPTO '97*, Lecture Notes in Computer Science Vol. 1294, B. Kaliski ed., Springer-Verlag, 1997.

[7] M. Bellare, O. Goldreich and S. Goldwasser, "Incremental cryptography with application to virus protection," Proc. 27th Annual Symposium on the Theory of Computing, ACM, 1995.

[8] M. Bellare, R. Guérin and P. Rogaway, "XOR MACs: New methods for message authentication using finite pseudorandom functions," *Advances in Cryptology – CRYPTO '95*, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[9] L. Carter and M. Wegman, "Universal Classes of Hash Functions," *Journal of Computer and System Science*, Vol. 18, 1979, pp. 143–154.

[10] I. Damgård, "A Design Principle for Hash Functions," *Advances in Cryptology – CRYPTO '89*, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989.

[11] O. Goldreich, S. Goldwasser and S. Micali, "How to construct random functions," *Journal of the ACM,* Vol. 33, No. 4, 210–217, (1986).

[12] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Science*, Vol. 28, 1984, pp. 270–299.

[13] S. Goldwasser, S. Micali and R. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal of Computing,* Vol. 17, No. 2, pp. 281–308, April 1988.

[14] M. Luby and C. Rackoff, "How to Construct Pseudorandom Permutations from Pseudorandom Functions," *SIAM Journal of Computing,* Vol. 17, No. 2, pp. 373–386, April 1988.

[15] R. Merkle, "One way hash functions and DES," *Advances in Cryptology – CRYPTO '89*, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989.

[16] R. Merkle, "A certified digital signature," *Advances in Cryptology – CRYPTO '89*, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989.

[17] M. Naor and M. Yung, "Universal one-way hash functions and their cryptographic applications," *Proceedings of the 21st Annual Symposium on the Theory of Computing*, ACM, 1989.

[18] E. Petrank and C. Rackoff, CBC MAC for real time data sources. *DIMACS Technical Report* 97-26, 1997.

[19] B. Preneel and P. van Oorschot, "MD-x MAC and building fast MACs from hash functions," *Advances in Cryptology – CRYPTO '95*, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[20] R. Rivest, "The MD5 message-digest algorithm," IETF RFC 1321 (April 1992).

[21] FIPS 180-1. Secure Hash Standard. Federal Information Processing Standard (FIPS), Publication 180-1, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., April 1995.

[22] M. Wegman and L. Carter, "New hash functions and their use in authentication and set equality," *Journal of Computer and System Sciences*, Vol. 22, 1981, pp. 265–279.

A Proofs of Lemmas

A.1 Proof of Lemma 4.2

Let A_N be a forger achieving the best possible success in attacking N, meaning it has resources at most t, q, μ and success $\mathbf{Succ}^{\mathrm{mac}}_N(A_N)$ equal to $\mathbf{InSec}^{\mathrm{mac}}_N(t, q, \mu)$. We will construct two adversaries, A_f and A_h, the first a forger attacking the MAC f and the second a weak collision-finder attacking h, so that the resources used by A_f will be at most $t, q, q(b + \ell)$ while those used by A_h will be at most t, q, μ, and furthermore

$$\mathbf{Succ}^{\mathrm{mac}}_f(A_f) + \mathbf{Succ}^{\mathrm{wcr}}_h(A_h) \geq \mathbf{Succ}^{\mathrm{mac}}_N(A_N) \ . \tag{8}$$

Thus we have

$$\begin{aligned} \mathbf{InSec}^{\mathrm{mac}}_N(t, q, \mu) &= \mathbf{Succ}^{\mathrm{mac}}_N(A_N) \\ &\leq \mathbf{Succ}^{\mathrm{mac}}_f(A_f) + \mathbf{Succ}^{\mathrm{wcr}}_h(A_h) \\ &\leq \mathbf{InSec}^{\mathrm{mac}}_f(t, q, q(b + \ell)) + \mathbf{InSec}^{\mathrm{wcr}}_h(t, q, \mu) \ , \end{aligned}$$

and Lemma 4.2 follows. The proof therefore reduces to presenting A_f and A_h to achieve Equation (8) while using the claimed resources.

Algorithms A_f and A_h will both use A_N as a subroutine, themselves providing answers to the oracle queries of A_N. Algorithm A_f has an oracle for $f(k_2, \cdot)$ where k_2 is a random key, and is trying to output a f-forgery, as per experiment $\mathrm{Forge}(A_f, f)$. In order to simulate the oracle $f(k_2, h(k_1, \cdot))$ that A_N expects to get, A_f will pick k_1 at random, compute $h(k_1, \cdot)$ itself, and then use its oracle to compute $f(k_2, \cdot)$ on the outcome. Algorithm A_h has an oracle for $h(k_1, \cdot)$ where k_1 is a random key, and is trying to find a h-collision, as per experiment $\mathrm{FindWeakCol}(A_h, h)$. In order to simulate the oracle $f(k_2, h(k_1, \cdot))$ that A_N expects to get, A_h will pick k_2 at random, use its oracle to compute $h(k_1, \cdot)$, and then compute $f(k_2, \cdot)$ itself on the outcome. The two algorithms are depicted in full below. We let q_N denote the number of oracle queries made directly by A_N.

Algorithm $A_f^{f(k_2,\cdot)}$
- Choose $k_1 \xleftarrow{R} \{0,1\}^\kappa$
- For $i = 1, \ldots, q_N$ do
  - $A_N \to x_i$
  - $A_N \leftarrow f(k_2, h(k_1, x_i))$
- $A_N \to (x, y)$
- $m \leftarrow h(k_1, x)$
- Return (m, y)

Algorithm $A_h^{h(k_1,\cdot)}$
- Choose $k_2 \xleftarrow{R} \{0,1\}^\kappa$
- For $i = 1, \ldots, q_N$ do
  - $A_N \to x_i$
  - $A_N \leftarrow f(k_2, h(k_1, x_i))$
- $A_N \to (x, y)$
- If there exists $j \in \{1, \ldots, q_N\}$ such that $h(k_1, x_j) = h(k_1, x)$
  - Return (x, x_j)

The algorithm A_f is the same as in [2]. The algorithm A_h was not explicitly given in [2] and is added to obtain the full proof of the concrete security claim we are making. Although the claims are essentially the same, we take a slightly different approach to the analysis than in [2].

Towards establishing Equation (8) we consider some events in the experiment $\mathrm{Forge}(A_N, N)$. Namely "A_N Succeeds" denotes the event that this experiment returns 1, and E denotes the event that for the string x output by A_N we have $h(k_1, x) \notin \{h(k_1, x_1), \ldots, h(k_1, x_{q_N})\}$. Below $\Pr[\,\cdot\,]$ refers to the probability under experiment $\mathrm{Forge}(A_N, N)$.

Adversary A_f succeeds in an f-forgery if A_N outputs a correct N-forgery for a message $x \notin \{x_1, \ldots, x_{q_N}\}$ (the event "A_N Succeeds") and $h(k_1, x) \notin \{h(k_1, x_1), \ldots, h(k_1, x_{q_N})\}$ (the event E). Thus

$$\mathbf{Succ}_f^{\mathrm{mac}}(A_f) = \Pr[\, A_N \text{ succeeds} \wedge E \,] \ . \tag{9}$$

However, if $x \notin \{x_1, \ldots, x_{q_N}\}$ but $h(k_1, x) = h(k_1, x_i)$ for some $i \in \{1, \ldots, q_N\}$ (namely $\overline{E}$ happens) then (x, x_i) is a collision pair for $h(k_1, \cdot)$ and the algorithm A_h will succeed in outputting a collision pair. (Notice that for A_h to output a correct collision pair for h, the success of A_N is not required; that is, even if A_N outputs an incorrect forgery, A_h might succeed.) Hence

$$\mathbf{Succ}_h^{\mathrm{wcr}}(A_h) \geq \Pr\left[\, A_N \text{ succeeds} \wedge \overline{E} \,\right] \ . \tag{10}$$

Using Equations (9) and (10) we have

$$\begin{aligned} \mathbf{Succ}_N^{\mathrm{mac}}(A_N) &= \Pr[\, A_N \text{ succeeds} \,] \\ &= \Pr[\, A_N \text{ succeeds} \wedge E \,] + \Pr[A_N \text{ succeeds} \wedge \overline{E}] \\ &\leq \mathbf{Succ}_f^{\mathrm{mac}}(A_f) + \mathbf{Succ}_h^{\mathrm{wcr}}(A_h) \end{aligned}$$

which yields Equation (8) as desired.

It remains to justify the claims about the resource parameters used by A_f and A_h. Recall that the resources pertain to the experiment rather than the adversary itself. Consider the experiments $\mathrm{Forge}(A_N, N)$, $\mathrm{Forge}(A_f, f)$ and $\mathrm{FindWeakCol}(A_h, h)$. Each experiment can be divided into three parts: generating a key for the oracle, running the adversary answering the oracle queries, and verifying the output given by the adversary. In all three experiments, the adversary A_N is eventually run either directly in the experiment itself or indirectly within the other adversaries A_f and A_h. Hence, within each experiment, the answers to the queries of A_N are computed either directly by the experiment (in case of $\mathrm{Forge}(A_N, N)$) or indirectly — partly by the experiment and partly by the adversary A_f or A_h (in $\mathrm{Forge}(A_f, f)$ and $\mathrm{FindWeakCol}(A_h, h)$ respectively.) The keys that are generated in each experiment including the keys generated by the adversaries are the key for the function N. This indicates that the key generation time and the oracle computation time are the same in each experiment. What seems to be different comes after the adversary A_N outputs a possible forgery (x, y). Within the algorithms for the adversaries A_f and A_h, it seems that extra computations need to take place to compute the output. Specifically, the computation of $m = h(k_1, x)$ is performed in both algorithms, and in A_h, finding x_j requires extra comparisons. However, if we consider things in the context of the experiments, essentially the same operations (ie. the computation of the functions based on the output of A_N and whether it was queried before) are performed either within the adversaries themselves or during their verification processes. (To be precise, there might be some minor differences in the sizes of the query sets and the functions that are computed in the verification process. However, these can be ignored especially since we are not considering the exact running time but an upper bound on the running time). Hence, we conclude that the actual running times of the three experiments are essentially equal.

To get what we called the running time we must also add in the size of the code of the adversaries in question. Adversaries A_f, A_h put small, constant size "wrappers" around the code of A_N and hence their code size is that of A_N plus $O(1)$. We have for simplicity ignored this minor additive constant in the time complexity since it is insignificant in practice.

To compute the number of queries for each experiment, we consider the query set (the set of all oracle queries of the adversary union with the adversary's message output) associated with each

experiment. For Forge(A_N, N), the query set is $Q = \{x_1, \ldots, x_{q_N}, x\}$. For Forge$(A_f, f)$, the query set is $Q_f = \{h(k_1, x_1), \ldots, h(k_1, x_{q_N}), h(k_1, x)\}$ since the queries are to the oracle $f(k_2, \cdot)$. For FindWeakCol(A_h, h), the query set is $Q_h = \{x_1, \ldots, x_{q_N}, x\}$ since the queried oracle $h(k_1, \cdot)$ takes the same input as the oracle $N_{k_1 k_2}(\cdot)$. Notice that x_j (the second part of the output pair) output by A_h is already included in the set Q_h since it was chosen among the already queried strings. Notice that $|Q_f| \leq |Q| = |Q_h|$. By assumption $|Q| \leq q$ so the query complexity of A_f and A_h is also at most q, as claimed. Since $Q_h = Q$ the total message complexity of A_h is the same as that of A, meaning at most μ. The total message complexity of A_f is $|Q_f| \cdot (b + \ell) \leq q(b + \ell)$ since the length of each query to the oracle $f(k_2, \cdot)$ is exactly $b + \ell$ bits.

A.2 Proof of Lemma 4.3

Let C_I be a collision-finder achieving the best possible success in attacking IT$[f]$, meaning it has resources at most t, q, μ and success $\mathbf{Succ}^{\mathrm{wcr}}_{\mathrm{IT}[f]}(C_I)$ equal to $\mathbf{InSec}^{\mathrm{wcr}}_{\mathrm{IT}[f]}(t, q, \mu)$. We will construct a collision-finder C_f attacking f so that the resources used by C_f will be at most t', q', μ' —where $t' = t$, $q' = \mu/b$ and $\mu' = (b + \ell)\mu/b$— and furthermore

$$\mathbf{Succ}^{\mathrm{wcr}}_{f}(C_f) \geq \mathbf{Succ}^{\mathrm{wcr}}_{\mathrm{IT}[f]}(C_I) \; . \tag{11}$$

Thus we have

$$\begin{aligned} \mathbf{InSec}^{\mathrm{wcr}}_{\mathrm{IT}[f]}(t, q, \mu) &= \mathbf{Succ}^{\mathrm{wcr}}_{\mathrm{IT}[f]}(C_I) \\ &\leq \mathbf{Succ}^{\mathrm{wcr}}_{f}(C_f) \\ &\leq \mathbf{InSec}^{\mathrm{wcr}}_{f}(t', q', \mu') \; , \end{aligned}$$

and Lemma 4.3 follows. The proof therefore reduces to presenting C_f to achieve the above claims.

For notational convenience, let IT$^*[f](k, x)$ denote the iterated function not including the concatenation of $\langle |x| \rangle$, namely:

Algorithm IT$^*[f](k, x)$
  $y_0 \leftarrow 0^\ell$
  Break x into b-bit blocks, $x = x_1 \ldots x_n$
  For $i = 1, \ldots, n$ do $y_i \leftarrow f(k, y_{i-1} \| x_i)$
  Return y_n

Thus IT$[f](k, x) =$ IT$^*[f](k, x) \| \langle |x| \rangle$.

The collision-finder C_f attacking f is presented in Figure 3. Here we let $X[j]$ denote the j-th b-bit block of a string X. We assume that queries and outputs of C_I have length a multiple of the block length b, since appropriate padding is used to ensure this anyway. Our algorithm C_f has an oracle for $f(k, \cdot)$, as per the definition of weak collision resistance. It begins by running C_I. The latter makes oracle queries to IT$[f](k, \cdot)$ (again, as per the definition of weak collision resistance) to which C_f provides the replies by itself computing IT$[f](k, \cdot)$. It can do the latter using its own oracle $f(k, \cdot)$ in the manner depicted in the subroutine ComputeIT$^{f(k,\cdot)}(\cdot)$. Accordingly C_f answers all the oracle queries of C_I, and then obtains the output (X, X') of C_I. Now it will use X, X' to try to find a collision in $f(k, \cdot)$, and will succeed whenever X, X' was a collision for IT$[f](k, \cdot)$.

Algorithm $C_f^{f(k,\cdot)}$

$i \leftarrow 0$
Repeat
$i \leftarrow i + 1$
$C_I \rightarrow X_i$
$C_I \leftarrow \text{ComputeIT}^{f(k,\cdot)}(X_i)$
Until C_I is done querying
$C_I \rightarrow (X, X')$
If $|X| \neq |X'|$ then Return Fail
Else
$y[0] \leftarrow 0^\ell$; $y'[0] \leftarrow 0^\ell$; $n \leftarrow |X|/b$; $i \leftarrow 0$
Repeat
$i \leftarrow i + 1$
$y[i] \leftarrow f(k, y[i-1] \| X[i])$
$y'[i] \leftarrow f(k, y'[i-1] \| X'[i])$
Until $(i \geq n)$ or $(y[i] = y'[i]$ and $(y[i-1] \| X[i] \neq (y'[i-1] \| X'[i]))$
Return $(y[i-1] \| X[i], y'[i-1] \| X'[i])$

Subroutine $\text{ComputeIT}^{f(k,\cdot)}(x)$

$y_0 \leftarrow 0^\ell$
Break x into b-bit blocks, $x = x_1 \ldots x_n$
For $i = 1, \ldots, n$ do $y_i \leftarrow f(k, y_{i-1} \| x_i)$
$s \leftarrow y_n \| \langle |x| \rangle$
Return s

Figure 3: Algorithms for proof of Lemma 4.3.

First note that if $|X| \neq |X'|$ then certainly $\text{IT}[f](k, X) \neq \text{IT}[f](k, X')$. This is because the last b bits of $\text{IT}[f](k, X)$ equal $\langle |X| \rangle$ while the last b bits of $\text{IT}[f](k, X')$ equal $\langle |X'| \rangle$, and the lengths are different by assumption. Accordingly if $|X| \neq |X'|$ then C_f returns Fail. We continue the analysis under the assumption that $|X| = |X'|$, and denote this common value by n.

By assumption n is a multiple of the block length b and $X = X[1] \ldots X[n]$ and $X' = X'[1] \ldots X'[n]$. We now show that the Repeat loop of C_f finds a collision in $f(k, \cdot)$ under the assumption that X, X' is a collision in $\text{IT}[f](k, \cdot)$. The quantities referred to in the following claim are as defined in the code for the algorithm C_f of Figure 3.

Claim A.1 Suppose $\text{IT}^*[f](k, X) = \text{IT}^*[f](k, X')$ but $X \neq X'$, where $|X| = |X'|$. Let $n = |X|/b$. Then there exists $i \in \{1, \ldots, n\}$ such that $y[i-1] \| X[i] \neq y'[i-1] \| X'[i]$ but $f(k, y[i-1] \| X[i]) = f(k, y'[i-1] \| X'[i])$.

Proof: The argument is by induction on the length n of the messages. The base case of the induction is when $n = 1$. In this case $\text{IT}^*[f](k, X) = f(k, y[0] \| X[1])$ and $\text{IT}^*[f](k, X') = f(k, y'[0] \| X'[1])$, where $y[0] = y'[0] = 0^\ell$. Thus our assumptions say that $f(k, y[0] \| X[1]) = f(k, y][0] \| X'[1])$ but $X[1] \neq X'[1]$, meaning $(y_0 \| X[1], y_0 \| X'[1])$ is a collision for $f(k, \cdot)$. In other words the claim is true with $i = 1$.

Now assume $n > 1$. The induction hypothesis is that the claim is true for $n-1$. We wish to establish the claim for n. The assumption is that $\mathrm{IT}^*[f](k, X) = \mathrm{IT}^*[f](k, X')$ but $X \neq X'$, and $n = |X|/b = |X'|/b$. By definition of $\mathrm{IT}^*[f](k, \cdot)$ we have

$$\begin{aligned} \mathrm{IT}^*[f](k, X[1]\cdots X[n]) &= f(k, \underbrace{\mathrm{IT}^*[f](k, X[1]\cdots X[n-1])}_{y[n-1]} \| X[n]) \\ \mathrm{IT}^*[f](k, X'[1]\cdots X'[n]) &= f(k, \underbrace{\mathrm{IT}^*[f](k, X'[1]\cdots X'[n-1])}_{y'[n-1]} \| X'[n])\ . \end{aligned}$$

Now consider two cases.

Case 1: $y[n-1]\|X[n] \neq y'[n-1]\|X'[n]$.

In this case, the claim is true with $i = n$. We did not even need to use the induction hypothesis.

Case 2: $y[n-1]\|X[n] = y'[n-1]\|X'[n]$.

This means in particular that $y[n-1] = y'[n-1]$. Furthermore, since $X[n] = X'[n]$ but $X \neq X'$ it must be that $X[1]\cdots X[n-1] \neq X'[1]\cdots X'[n-1]$. Thus, the induction hypothesis tells us that there is an $i \in \{1, \ldots, n-1\}$ for which the claim is true. ∎

This completes the proof of Equation (11). We now analyze the resource parameters in the context of the experiments $\mathrm{FindWeakCol}(C_I, \mathrm{IT}[(]f))$ and $\mathrm{FindWeakCol}(C_f, f)$. Each experiment can be divided into three parts: key generation, computation for answers to oracle queries, and output verification. In $\mathrm{FindWeakCol}(C_I, \mathrm{IT}[f])$, algorithm C_I is run directly while in $\mathrm{FindWeakCol}(C_f, f)$, it is run indirectly within the adversary C_f. Since the two experiments run the same adversary C_I, they have the same running time for the first two parts (key generation and computation of oracle queries) up until C_I outputs its string pair. After C_I outputs the pair of strings, the experiment $\mathrm{FindWeakCol}(C_I, \mathrm{IT}[f])$ verifies whether they are a collision pair by computing the function $\mathrm{IT}[f](k, \cdot)$ on the output strings and comparing the results. This process roughly corresponds to the second loop in the algorithm for C_f, where it finds a collision pair for the function $f(k, \cdot)$ by computing the function $f(k, \cdot)$ of each block in the output strings of C_I and comparing the results until it finds a collision pair for $f(k, \cdot)$. Although there are some minor difference in total lengths of strings compared —the total length of strings compared in $\mathrm{FindWeakCol}(C_f, f)$ is slightly larger than that in $\mathrm{FindWeakCol}(C_I, \mathrm{IT}[f])$— if we assume the time for comparison is small enough, it is reasonable to ignore this difference. (This would hardly be an invalid assumption, since in general, compared to the time for the rest of the operations in either experiment, the time for the comparison of the final output strings would be negligible.) This tells us that the execution times are pretty much the same. Finally we must add in the the size of the code of the algorithms. That of C_f is slightly more than that of C_I, but again we decide to ignore this slight difference. Upto a negligible difference, we thus claim that $t' = t$.

Regarding the number of queries q' made to oracle $f(k, \cdot)$, notice that it may not be easily expressed in terms of q since each query to the oracle $\mathrm{IT}[f](k, \cdot)$ may take a variable number of queries to oracle $f(k, \cdot)$. Hence, we express q' in terms of μ (the sum of the lengths of all messages queried and output by C_I). To compute the number of queries to $f(k, \cdot)$, we need to compute the total number of blocks in μ since each block in the queries for $\mathrm{IT}[f]_k$ corresponds to the oracle query for $f(k, \cdot)$. The number of blocks in μ is obtained by dividing it by the block length b. Notice that the final output message pair of C_I may be only partly processed by C_f since it stops once a collision pair is found for $f(k, \cdot)$. Since q', by definition, is an upper-bound, $q' = \mu/b$.

The sum of lengths of all queries of C_f (μ') can be computed by multiplying the number of queries by the length of each query. Since the length of each query for the oracle $f(k,\cdot)$ is $b+\ell$ and the number of queries is at most μ/b, we have $\mu' = (b+\ell)\mu/b$.

A.3 Proof of Lemma 4.4

Let C be a collision-finder achieving the best possible success in attacking g, meaning it has resources at most t, q, μ and success $\mathbf{Succ}^{\mathrm{wcr}}_g(C)$ equal to $\mathbf{InSec}^{\mathrm{wcr}}_g(t,q,\mu)$. We will construct a forger A attacking g, so that the resources used by A will be at most t', q', μ' —where $t' = t + O(\mu)$, $q' = q$ and $\mu' = \mu$— and furthermore

$$\mathbf{Succ}^{\mathrm{mac}}_g(A) \geq \frac{2}{q(q-1)} \cdot \mathbf{Succ}^{\mathrm{wcr}}_g(C)\ . \tag{12}$$

Thus we have

$$\begin{aligned}
\mathbf{InSec}^{\mathrm{wcr}}_g(t,q,\mu) &= \mathbf{Succ}^{\mathrm{wcr}}_g(C) \\
&\leq \frac{q(q-1)}{2} \cdot \mathbf{Succ}^{\mathrm{mac}}_g(A) \\
&\leq \frac{q(q-1)}{2} \cdot \mathbf{InSec}^{\mathrm{mac}}_g(t',q',\mu')\ ,
\end{aligned}$$

and Lemma 4.4 follows. The proof therefore reduces to presenting A to achieve Equation (12) while using the claimed resources.

Algorithm A gets an oracle for $g(k,\cdot)$ so that it can mount a chosen-message attack. It runs C providing answers to C's queries using its own oracle. For simplicity assume that all the oracle queries made by C are distinct. (Since C's oracle is deterministic, there is no reason to repeat an oracle query.) Let $Q_C = \{x_1, \ldots, x_{q_C}\}$ denote the set of queries made by C and let (y_0, y_1) denote the pair of strings output by C after it has completed its interaction with its oracle. We assume $y_0 \neq y_1$ since otherwise C cannot be successful. The query set of C is $Q = Q_C \cup \{y_0, y_1\}$. By assumption it has size at most q, and we assume for simplicity it has size exactly q. We write $Q = \{x_1, \ldots, x_q\}$. This means we have assigned indices a, b such that $y_0 = x_a$ and $y_1 = x_b$.

Notice that either y_0 or y_1 or both or neither may be in Q_C: we do not know a priori, and it may vary from execution to execution. If, say, y_0 is already in Q_C, then $a \leq q_C$, and otherwise the range of the indices for the points in Q_C is extended to accomodate y_0. Similarly for y_1. This inclusion of the collision points and queried points in a common indexed set is convenient to what follows.

Algorithm A is presented in Figure 4. The idea is that A will pick two distinct points x_j, x_i at random from Q and hope that they form a collision for $g(k,\cdot)$. (It may be worth observing that $q \geq 2$ due to the presence of the two assumed distinct points y_0, y_1 in the query set, so that the random choices of algorithm A are valid.) If so we would have $g(k,x_j) = g(k,x_i)$. Then A will output $(x_i, g(k,x_j))$ as a forgery. There is one catch: this is only valid if x_i was not an oracle query of A. This requires, first, that $x_i \neq x_j$, but that is true because the points in Q are distinct by assumption. However, x_i might have been an oracle query of C, and in the process of answering the oracle queries of C, algorithm A would have answered this query, meaning would have queried its own oracle at x_i. To make sure that A did not make oracle query x_i, the execution of C is halted just after it outputs its i-th query and just before that query is answered. This is reflected in the code of A in the query reply stage, where the answer to oracle query x_s of C is only provided if $s \leq i-1$.

Algorithm $A^{g(k,\cdot)}$

Choose $i \xleftarrow{R} \{2,\ldots,q\}$; $\quad j \xleftarrow{R} \{1,\ldots,i-1\}$
$s \leftarrow 0$
While $(s \leq i-1)$ and (C has not finished querying) do
  $s \leftarrow s+1$
  $C \rightarrow x_s$
  If $s \leq i-1$ then $C \leftarrow g(k, x_s)$
End While
If (All queries of C have been answered) then
  $C \rightarrow (y_0, y_1)$
  If $y_0 \notin \{x_1,\ldots,x_s\}$ then $s \leftarrow s+1$; $x_s \leftarrow y_0$ End If
  If $y_1 \notin \{x_1,\ldots,x_s\}$ then $s \leftarrow s+1$; $x_s \leftarrow y_1$ End If
End If
Return $(x_i, g(k, x_j))$

Figure 4: Forger A for proof of Lemma 4.4.

However, remember that $\{x_1,\ldots,x_q\}$ includes by definition the points y_0, y_1 that are output by C. In particular, the values of i or j might refer to these points. Accordingly the last part of the code of A makes sure that these points are assigned indices in the sense that each has the form x_m for some m. If the point was already queried, its index has been assigned and there is nothing to do, but otherwise the running index s for the set of points must be incremented and the value x_s assigned to the new point. Notice that this code is only executed if all queries of C have been answered (not just asked, but answered); indeed, otherwise it is not even possible to continue running C. This means in particular that this code is executed when the chosen value of i is strictly more than the number of oracle queries actually made by C.

Even if y_0, y_1 are not obtained by A because it did not reach that part of its code, we will refer to them in the correctness argument below. They are simply the values that C would have output had A continued to run it. These values are well-defined because C's coins (if any) have been (initially chosen at random and then) fixed by A.

Now let α be the (unique) value in $\{1,\ldots,q\}$ such that $y_0 = x_\alpha$ and let β be the (unique) value in $\{1,\ldots,q\}$ such that $y_1 = x_\beta$. Let $a = \max(\alpha,\beta)$ and $b = \min(\alpha,\beta)$. Notice that $1 \leq b < a \leq q$ and if C is successful then (x_b, x_a) is a collision for $g(k,\cdot)$. Let E be the event that $(i,j) = (a,b)$. We claim that if E happens then A is successful in finding a forgery for $g(k,\cdot)$. This requires checking two things: that $g(k,x_j)$ is a valid tag for x_i, and that x_i was not queried by A. Both are relatively easy to see. The first is true because (x_i, x_j) is a collision for $g(k,\cdot)$. The second is true because the code of A makes sure it does not query x_i. Now since the number of choices for (a,b) is $q(q-1)/2$ and i,j are chosen at random in the manner indicated in the code, we have Equation (12) as desired.

The relationship between the resource parameters can be obtained as usual by comparing the maximum resources used by A to those used by C in their respective experiments. Regarding the running time, A and C basically have the same running time except for the "If" statement of A that follows the "While" loop. That takes $O(\mu)$ additional time, where μ is the sum of lengths of all queries of C. Also, in the context of the experiments $\mathrm{Forge}(A,g)$ and $\mathrm{FindWeakCol}(C,g)$, we need take the verification time into consideration. The difference in verification mainly lies in that

the verification of a forgery requires checking the newness of the message whereas that of a collision does not. Checking whether the output message of the forger is new with respect to its previous queries takes $O(\mu)$. Hence, $t' = t + O(\mu)$.

To compute the number of queries, we consider the query set. In $\mathrm{Forge}(A, g)$, the query set is (a subset of) $\{x_1, \ldots, x_q\}$. Both strings y_0, y_1 are included in the set for $\mathrm{Forge}(A, g)$, since one of them is the output string of A and the other one is queried by A. Hence, $q = q'$ and $\mu = \mu'$.