Message Authentication Codes from Unpredictable Block Ciphers

Yevgeniy Dodis* John Steinberger†

June 2, 2009

Abstract

We design an efficient mode of operation on block ciphers, SS-NMAC. Our mode has the following properties, when instantiated with a block cipher f to yield a variable-length, keyed hash function H:

(1) **MAC Preservation.** H is a secure message authentication code (MAC) *with birthday security*, as long as f is *unpredictable*.

(2) **PRF Preservation.** H is a secure pseudorandom function (PRF) *with birthday security*, as long as f is *pseudorandom*.

(3) **Security against Side-Channels.** As long as the block cipher f does not leak side-channel information about its internals to the attacker, properties (1) and (2) hold even if the remaining implementation of H is *completely leaky*. In particular, if the attacker can learn the transcript of all block cipher calls and other auxiliary information needed to implement our mode of operation.

Our mode is the first to satisfy the MAC preservation property (1) with birthday security, solving the main open problem of Dodis et al. [8] from Eurocrypt 2008. Combined with the PRF preservation (2), our mode provides a hedge against the case when the block cipher f is more secure as a MAC than as a PRF: if it is false, as we hope, we get a secure variable-length PRF; however, even if true, we still "salvage" a secure MAC, which might be enough for a given application.

We also remark that no prior mode of operation offered birthday security against side channel attacks, even if the block cipher was assumed pseudorandom.

Although very efficient, our mode is three times slower than many of the prior modes, such as CBC, which do not enjoy properties (1) and (3). Thus, our work motivates further research to understand the gap between unpredictability and pseudorandomness of the existing block ciphers, such as AES.

*Computer Science Dept. NYU. Email: dodis@cs.nyu.edu.

†Department of Mathematics, University of British Columbia, Email: jpsteinb@gmail.com.

1 Introduction

Most primitives in symmetric-key cryptography are built from block ciphers, such as AES. Typically, one models the block cipher as a fixed-input-length (FIL) *pseudorandom* permutation (PRP), and then builds a more complex variable-input-length (VIL) primitive under this assumption. For many such VIL primitives, like pseudorandom functions (PRFs), this strong assumption on the block cipher is justifiable. One exception here is the design of message authentication codes (MACs): since the resulting primitive only needs to be *unpredictable*, it would be highly desirable to only assume that the block cipher is unpredictable as well, as opposed to pseudorandom. Indeed, it seems that assuming the block cipher is unpredictable is a *much weaker* assumption than assuming full pseudorandomness: to break the latter, all one needs to do is to predict one bit of "random-looking" information about the block cipher with probability just a little over 1/2, while the former requires one to fully compute the value of the block cipher on a new point with non-trivial probability. Thus, it is natural to ask the following central question of this work:

Question 1 *Can one build an efficient variable-input-length MAC from a block cipher which is modeled as an* unpredictable *permutation (UP) on n-bits?*

We will argue that no existing constructions are really satisfactory for achieving this natural goal. In order to discuss this precisely, we briefly recall some key quantities which determine the security of a construction. In this paper we consider only two types of adversaries: *distinguishers*, whose goal is to distinguish a function from an ideal primitive (as for PRFs and PRPs) and *forgers*, whose goal is to predict the value of the function on an un-queried message (as for MACs and UPs). As there often exist constant-query attacks using very long messages, the most important measure of an adversary's efficiency is the total length of messages that it queries. This upper bounds, among others, the number of queries made by the adversary. For functions that are built from a smaller primitive (such as, in all the cases we consider, a permutation), a more convenient efficiency measure is the number of queries one must make to the smaller primitive in order to evaluate the adversary's queries. In this section we let q stand for the latter number, as opposed to the number of queries actually made by the adversary to its oracle.

Let C be a function built from a block cipher f. The *security*[1] $\varepsilon = \varepsilon(q)$ of C is the maximum advantage of an adversary for which the number of calls to f necessary to compute the adversary's queries to C does not exceed q. Thus, lower ε implies better security. We write ε_{mac} and ε_{prf} to distinguish the security C as MAC and PRF, respectively. Likewise the block cipher has a security ε_{up} and ε_{prp} as a UP and as PRP, respectively.[2] The *rate* of a VIL-function C is the number of times the block cipher has to be called on each input block, so it measures the efficiency of C. We can now rephrase our goal as follows. Given a block cipher f with UP security ε_{up}, construct a VIL-MAC C such that:

(a) C has small constant rate;

(b) the security ε_{mac} of C is as small as possible as a function of ε_{up}.

A good way to quantify the "goodness" of ε_{mac} is to assess the maximum q for which the achieved bound is meaningful, assuming that the block cipher has ideal security $\varepsilon_{\text{up}} \sim 1/2^n$ as an unpredictable permutation. For example, if $\varepsilon_{\text{mac}} = O(\varepsilon_{\text{up}} \cdot q^2)$, then the bound is meaningful for q up to $\sim 2^{n/2}$, which matches the classical birthday bound typically achieved when one models the block cipher as a PRP. As

[1] Other parameters, such as the running time allowed to the adversary, may be relevant for the security, but these are not important here.

[2] In fact $\varepsilon_{\text{up}} = \varepsilon_{\text{mac}}$ for a block cipher, since these refer to the same notion, but we write ε_{up} to emphasize the difference with C.

argued by Preneel and van Oorschot [17], a simple extension attack shows that the birthday security is the best security one may hope to achieve by natural "iterative" constructions. On a positive, several elegant (iterative) constructions matching this bound are known, when modeling the block cipher as a PRP. On a negative, no existing constructions, iterative or otherwise, come even close to the birthday security when assuming UPs as opposed to PRPs. Thus, our "golden standard" to answer Question 1 will be to solve

Question 2 *Build an (iterative) VIL-MAC from UPs, having constant efficiency rate and (nearly) birthday security.*

Jumping ahead, this will be our main result, therefore resolving the main open question of Dodis et al. [8] from Eurocrypt 2008. But first, let us survey what is known, to better understand the difficulties we will have to face, and also motivate our approach.

1.1 Inapplicability of Existing Solutions

There is a huge number of proposals for building a VIL-MAC out of a block cipher. Unfortunately, it turns out that most of them are insecure when instantiated with unpredictable block ciphers, — often despite having simple proofs of security when one models the block cipher as a PRP, — and the few that are secure, achieve extremely poor rate and/or security. In order to keep the present discussion focused, we give a detailed listing of many "failed" approaches in Appendix A, here concentrating only on the highlights relevant to our actual approach.

In brief, the existing approaches in question include the following: (1) generic route from unpredictability to pseudorandomness [11, 14]; (2) CBC-MAC [4, 15]; (3) HMAC/NMAC [3, 6]; (4) various ad-hoc methods (e.g., iterating the truncated version of the block cipher); (5) hash-then-MAC using (almost) universal hashing [3, 5]; (6) hash-then-MAC using collision-resistant hashing; (7) Feistel Network [9, 12]; and (8) the current best method called "enhanced CBC" mode [8]. Of these, approaches (2), (3), (4) and (5) are completely insecure when instantiated with generic UPs (as opposed to PRPs!). This is simple to see for (3), (4) and (5), and was shown by An and Bellare [2] for the CBC-MAC (approach (2)). The generic approach (1) is secure, but very inefficient, which is not surprising.

Approach (6), using a collision-resistant hash function (CRHF) H to hash the VIL message before applying a FIL-MAC, also works in principle, but is not satisfactory. In theory, the assumption that CRHFs exist is much stronger than the existence of UPs (or even PRPs); for example, there is a black-box separation [20] between these assumptions. Even in practice, where many hash functions are built from block ciphers (and analyzed in the ideal cipher model [7, 16]), the resulting hash functions appear to require a larger security parameter than the "stand-alone" block ciphers they are built from. For example, while the industry standard AES has input length 128, no existing hash function with 128-bit output is considered secure (e.g., MD5 and related functions are broken [21]); in fact, NIST does not recommend using any hash function with output size below 256, including 160-bit SHA-1.

Weak Collision-Resistance. Thus, we would like to base security of the "hash-then-mac" approach on weaker hash functions than CRHFs. As was demonstrated by [6], the precisely correct notion for this task is that of *Weak Collision Resistance* (WCR). Such hash functions H are keyed, and their key is part of the secret key for the resulting VIL-MAC. In terms of security, it should be infeasible for the attacker to come up with distinct inputs x and y such that $H(x) = H(y)$, *even when given oracle access to* H. An and Bellare [2] then showed that WCR hash functions have similar properties to CRHFs: in particular, the (strengthened) Merkle-Damgard transform gives a VIL-WCR hash function from a FIL-WCR hash function, which can then be used in "hash-then-mac" approach. Moreover, both security reductions are tight for our purposes.

Thus, to efficiently answer Question 1, *it is sufficient to build a fixed-input-length sufficiently compressing (say, two-to-one) WCR hash family.* Indeed, this is the route of all existing solutions (e.g.,

approaches (7) and (8)), as well as our solution. However, to also answer Question 2 would additionally require a WCR hash *with birthday security*, which was not known prior to this work.

BUILDING WCR FROM UP. This question appears to be non-trivial. In fact, many of the failed approaches above, such as (2), (3), (4) and (5), not only fail to give a *shrinking* MAC already on two-block messages, but also fail to yield a WCR hash. (Note, a shrinking MAC is also WCR [2].) In particular, only two secure solutions were known prior to this work (approaches (7) and (8) above). First, Dodis and Puniya [9] showed how to construct a two-to-one FIL-WCR from $\omega(\log \kappa)$ independent UPs, where κ is the security parameter. The construction applied $\omega(\log \kappa)$ rounds of the Feistel Network $\pi(x\|y) = (y\|f(y) \oplus x)$ to the $2n$-bit input, each with a different UP f, and then truncated the last output in half. Moreover, they showed that $O(\log \kappa)$ rounds are generally insufficient for this task (extending the three-round counter-example of [2], and in sharp contrast to the setting of PRPs, where three rounds are already enough [12]). This means that the resulting super-constant rate $\omega(\log \kappa)$ of this particular construction cannot be improved, making it somewhat inefficient for practice. More significantly, the security of this construction proven by [9] was only $O(\varepsilon_{\text{up}} \cdot q^6)$, meaning that it can only be secure for at most $2^{n/6}$ messages, which is unacceptable for $n = 128$.

The best current WCR construction from UPs comes from the work of Dodis et al. [8], who made the surprisingly simple observation that the function $h(x\|y) = f_1(x) \oplus f_2(y)$ is a two-to-one, rate-2 WCR hash function, assuming f_1 and f_2 are two independent UPs. This immediately gives a rate-2 VIL-MAC from UPs, which is very efficient, and is the first (and only) constant-rate solution to Question 1 known prior to this work. Unfortunately, the security of this WCR function (and the resulting VIL-MAC) is $O(\varepsilon_{\text{up}} \cdot q^4)$. Moreover, it is easy to see that this bound is actually tight. Thus, the construction can only be secure for at most $2^{n/4}$ messages, again making it fall short of our goal of obtaining security up to $2^{n/2}$. In fact, this question of achieving "birthday security" $2^{n/2}$ (which is our Question 2) was the main open question posed in [8].

1.2 Our Results

In this work we resolve this question in the affirmative. Concretely, we construct a VIL-MAC, called SS-NMAC, from four independent UPs $f_1, \ldots, f_4$, which achieves rate 3 and security $\varepsilon_{\text{mac}} \approx O(\varepsilon_{\text{up}} q^2 \log^2(q))$, meaning it can be secure for almost $2^{n/2}$ messages. This is the first constant-rate MAC with birthday security built from an unpredictable block cipher. The construction of SS-NMAC is depicted in Figure 2, where the message is $x = x_1 \ldots x_\ell$.

Our construction uses the WCR approach mentioned earlier: namely, it uses the (strengthened) Merkle-Damgard iteration of the $2n$-bit to n-bit compression function $F(x\|y) = f_1(x) \oplus f_3(f_1(x) \oplus f_2(y))$, which is shown in Figure 1. This function was originally suggested by Shrimpton and Stam [19], who argued that F is collision-resistant with birthday security, assuming that f_1, f_2, f_3 are *public random functions* (i.e., random oracles). In contrast, our main technical result (Theorem 1) shows that this function is (weakly) collision-resistant (with birthday security), even if f_1, f_2, f_3 are only (keyed) *unpredictable* functions.

We note that since any FIL-MAC is FIL-WCR (Lemma 4.4 [2]) it would suffice to prove the Shrimpton-Stam compression function is a good MAC in order to show it is WCR. However, as explained in Appendix C, the Shrimpton-Stam compression function is not a good enough MAC for our purposes, showing the necessity of directly proving WCR security.

COMPARISON WITH [19]. On a technical level, both results appear similar. In both cases, assuming the adversary A has oracle access to f_1, f_2, f_3, one has to argue that A has low chance of finding a collision to F. However, the key difference is that the f_i's are assumed truly random in [19], whereas we can only assume *unpredictable* f_i's. In particular, while [19] could directly bound the probability of A finding the collision in F using an *information-theoretic argument*, we have to *build an efficient reduction* from the presumed collision-finding attacker A to a UP-forger B forging one of the MACs. It is well known that

such information-theoretic arguments often do not have direct analogs in the computational setting. To illustrate this more concretely, let us only discuss the most interesting such difficulty we had to resolve.

The key argument of [19] was a technical calculation, using factorials, binomials and various probability manipulations, that A is unlikely to find an "n-way multi-collision" in the auxiliary function $h(x\|y) = f_1(x) \oplus f_2(y)$, when f_1 and f_2 are truly random functions. To adapt this (critical) part of the argument to our computational setting, we would have to take an efficient attacker A' capable of finding such a multi-collision in h with probability ε, and turn it into a forger B' for either f_1 or f_2, succeeding with probability $\Omega(\varepsilon/q^2)$. As far as we could see, the probability calculations in [19] give no guidance of how to do such a reduction. And, indeed, finding such a reduction required a completely new approach, relating to a natural "balls-and-bins" game that we analyzed (see Lemma 1), and resulting in a very non-obvious construction of B. We discuss this construction in detail in Section 4.2, only mentioning that it gave us a better understanding of the security of the Shrimpton-Stam compression function, and even implicitly improved the probability calculations of [19] for the special case of truly random functions (corresponding to $\varepsilon = 2^{-n}$ in our reduction). In particular, for the case of random functions we get a convenient closed form of $O(q^2 \log^2(q)/2^n)$ for the collision resistance of the Shrimpton-Stam compression function, where, as per our convention for this section, q is the total number of block cipher queries allowed (cf. Theorem 1).

Strong PRF preservation. We also notice that our new mode has the following desirable multi-property preservation guarantee advocated by [8]: if the block cipher is unpredictable, we get a MAC with message security roughly $2^{n/2}$, while if it happens to be pseudorandom, we get a PRF with message security roughly $2^{n/2}$. In other words, we expect and hope that practical block ciphers (such as AES) are in fact PRPs with good security. If our hope is correct, we would get a full-fledged pseudorandom function with good security; however, even if the block cipher turns out to be a much better MAC than it is a pseudorandom function, we still get a MAC with excellent security, which could be reassuring for many applications. Details are sketched in Section 5.

More interestingly, even in the setting of PRPs, *our SS-NMAC construction yields a more "leakage-resilient" VIL-PRF H than the prior constructions.* In particular, in Section 6 we show that the resulting PRF is secure even in the so called *oracle cipher model*, first considered by Dodis et al. [9]. Recall, in the standard model for PRFs, the attacker only learns the output $H(x)$ of the PRF on input x, but does not learn any of the intermediate values, such as the inputs/outputs to the block cipher or any of the chaining variables. Indeed, this secrecy of the intermediate values is completely essential to the security of most standard constructions, such as CBC-MAC or the standard Luby-Rackoff transformation [12]. In other words, these constructions are actually *broken* in the oracle cipher model, irrespective of the strength of the block cipher used (e.g., even with AES). In contrast, our SS-NMAC construction is a secure VIL-PRF — *with (essentially) the same birthday security* — even when the attacker *learns all the intermediate values needed to obtain $H(x)$*, except for what is done inside the actual block-cipher computations. More precisely, even if the attacker learns all the computation history of our SS-NMAC construction on a bunch of points (not including the internals of the block ciphers), the value of the function at any set of *non-queried* points looks random to the attacker. Thus, as long as the block cipher is implemented in a "leakage-resilient" way, the remaining implementation of SS-NMAC can be completely insecure with respect to side-channel attacks! We believe that the security in the oracle model is quite important, since we envision secure *hardware-based* implementation of block-ciphers, later composed with much less secure *software-based* solutions, to yield various more advanced VIL primitives. We also remark that none of the two previous PRF constructions in the oracle cipher model [8, 9] achieved anything close to birthday security.

Summary. To summarize, in addition to yielding a more secure VIL-MAC than prior constructions in the case when $\varepsilon_{\text{up}} \ll \varepsilon_{\text{prp}}$, our construction gives a more "leakage-resilient" (and equally secure!) VIL-PRF even when assuming ε_{prp} is nearly as good as ε_{up}. Moreover, we only pay a small constant factor price in efficiency for these (significant) security enhancements. In Section 7, we briefly discuss

whether this slowdown is justifiable in practice, which ultimately calls for more research to understand the gap between unpredictability and pseudorandomness of the existing block ciphers, such as AES.

2 Security Definitions

We briefly recall the standard security notions for MACs and PRFs. In each case we are interested in resistance to chosen message attacks. For a MAC, an adversary succeeds if it can forge the MAC on an un-queried value. For a PRF, the adversary succeeds if it can distinguish the PRF from a truly random oracle. To measure an adversary's efficiency we count not only the number of oracle queries made but also the time and the total length of queried messages (as the oracles accept variable length inputs). In this section we use the variable $\tilde{q}$ to denote the number of queries made by the adversary to its oracle in order to emphasize the distinction from the variable q used in Section 1, which was defined as the (distinct) number of block cipher calls necessary to evaluate those queries (the adversary does not have direct access to the block cipher). In later sections we maintain the spirit of this convention, using $\tilde{q}$ for queries made to VIL-functions and q for queries made to FIL-functions (usually block ciphers).

A *function family* is a map $f : \{0,1\}^\kappa \times Dom(f) \to \{0,1\}^n$ where $Dom(f) \subseteq \{0,1\}^*$. The strings in $\{0,1\}^\kappa$ are the *keys* of f and we write $f_k(x)$ for $f(k,x)$ for $k \in \{0,1\}^\kappa$ and $x \in Dom(f)$. The function f_k is called a *member* of f.

For MACs we consider the following game, where A is an adversary with oracle access to f_k:

Game Forge(A, f):
$k \leftarrow \{0,1\}^\kappa$; $(x, y) \leftarrow A^{f_k}$
If $x \in Dom(f)$, $f_k(x) = y$ and x was not a query of A then A wins, otherwise A looses.

Following An and Bellare [2] we define the insecurity of f as a MAC to be

$$\mathbf{InSec}_f^{\mathrm{mac}}(t, \tilde{q}, \mu) := \max_A \Pr[A \text{ wins Forge}(A, f)]$$

where the maximum is taken over all adversaries A making at most $\tilde{q}$ queries whose total combined length is at most μ bits and of "running time" at most t. The "running time" is defined to be the total running time of the experiment, including the time necessary to compute the answers to A's queries. (The advantage of this definition is that a simulator running A and computing the answer to A's queries from scratch has essentially the same running time t.)

For PRF security the game is modified by either giving A access to a random f_k or to a random oracle $g : Dom(f) \to \{0,1\}^n$ with probability $\frac{1}{2}$ and A wins if it correctly identifies whether its oracle is f_k or g. Call this game 'Identifies(A, f)'. Then

$$\mathbf{InSec}_f^{\mathrm{prf}}(t, \tilde{q}, \mu) := \max_A \Pr[A \text{ wins Identifies}(A, f)] - \frac{1}{2}$$

where again the maximum is taken over all adversaries A making at most $\tilde{q}$ queries of total length μ and of running time t, with the same convention concerning running time.

The proof finally uses the notion of "weak collision resistance" (WCR), which measures the collision resistance of a function only available as an oracle to the adversary. In the weak collision resistance game for the function family f, A is given oracle access to a random f_k and wins if it succeeds in querying f_k on two distinct points x, y such that $f_k(x) = f_k(y)$. Then $\mathbf{InSec}_f^{\mathrm{wcr}}(t, \tilde{q}, \mu)$ is defined similarly with respect to this game as $\mathbf{InSec}_f^{\mathrm{mac}}(t, \tilde{q}, \mu)$ is defined with respect to the game Forge(A, f).

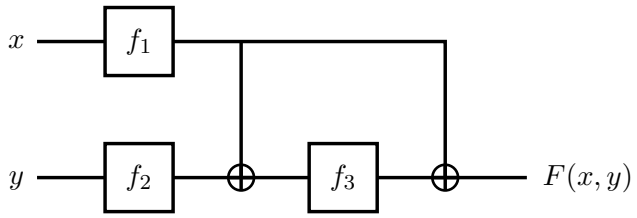

Figure 1: The Shrimpton-Stam compression function. All wires carry n-bit values.

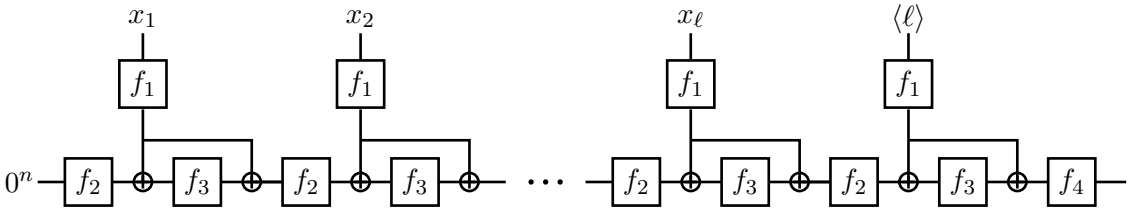

Figure 2: The SS-NMAC mode of operation.

3 The SS-NMAC construction

The basic SS-NMAC scheme is shown in Figure 2. The scheme uses the Merkle-Damgard iteration of the $2n$-bit to n-bit compression function of Shrimpton and Stam [19] shown in Figure 1. We start by describing this compression function.

The Shrimpton-Stam compression function. The Shrimpton-Stam compression function is a $2n$-bit to n-bit compression function that uses calls to three different n-bit to n-bit primitives f_1, f_2, f_3. We write the compression function as $F[f_1, f_2, f_3]$ to emphasize its dependence on f_1, f_2, f_3. It is defined by

$$F[f_1, f_2, f_3](x\|y) = f_1(x) \oplus f_3(f_1(x) \oplus f_2(y))$$

for any pair of n-bit strings x, y.

Shrimpton and Stam [19] proved that $F[f_1, f_2, f_3]$ has optimal (i.e. birthday) collision resistance if f_1, f_2, f_3 are random functions. They also conjectured that the construction remains collision resistant if $f_i(x)$ is replaced with $\pi_i(x) \oplus x$ where π_1, π_2, π_3 are random permutations, which would enable the construction to be implemented with fixed key block ciphers. This conjecture was verified by Rogaway and Steinberger [18].

For our purposes, the key property of $F[f_1, f_2, f_3]$ is that an adversary with oracle access to the f_i's can only learn $F[f_1, f_2, f_3](x\|y)$ on roughly as many inputs $x\|y$ as it makes queries. This should be contrasted for example to the compression function $h[f_1](x\|y) = f_1(x \oplus y)$ of the CBC MAC or the "xor compression function" $g[f_1, f_2](x\|y) = f_1(x) \oplus f_2(y)$ of the enciphered CBC construction of Dodis, Pietrzak and Puniya [8], where f_1, f_2 are again n-bit to n-bit functions. An adversary querying $h[f_1]$ can learn to evaluate $h[f_1]$ on 2^n inputs $x\|y$ in a single query; an adversary querying $g[f_1, f_2]$ can learn to evaluate $g[f_1, f_2]$ on q^2 inputs $x\|y$ in q queries. Another compression function that could be used equally well in place of $F[f_1, f_2, f_3]$ is the LP231 compression function of Rogaway and Steinberger [18], which also uses three calls to n-bit to n-bit primitives. However we use $F[f_1, f_2, f_3]$ because it is simpler and sufficient for our purposes.

Iteration and Padding. First we define PadAp(x) to be $x10^k\langle\ell\rangle$ where k is the least integer such that $x10^k$ has length a multiple of n, where ℓ is the number of n-bit blocks in $x10^k$, and where $\langle\ell\rangle$ is ℓ written as an n-bit binary integer (messages with maximum length 2^n are sufficient for most

applications). Appending $\langle \ell \rangle$ amounts to using Merkle-Damgard strengthening, which we do in order to keep our space of messages suffix-free. Any other suffix-free encoding of messages would do as well.

To iterate $F[f_1, f_2, f_3]$ we define the "SS-cascade" $G[f_1, f_2, f_3]$ of an $n\ell$-bit string $x = x_1 \| \cdots \| x_\ell$ where each x_i is an n-bit string by $G[f_1, f_2, f_3](x) = y_\ell$ where $y_0 = 0^n$ and $y_k = F[f_1, f_2, f_3](x_k \| y_{k-1})$ for $1 \leq k \leq \ell$. Finally, given an additional n-bit to n-bit function f_4 we define the SS-NMAC $H[f_1, f_2, f_3, f_4]$ by

$$H[f_1, f_2, f_3, f_4](x) = f_4(G[f_1, f_2, f_3](x)).$$

for all $x \in \mathrm{Dom}(H) := \{\mathrm{PadAp}(y) : y \in \{0,1\}^*\}$. See Figure 2, where $x = x_1 \| \cdots \| x_\ell \| \langle \ell \rangle$. Note that to query $H[f_1, f_2, f_3, f_4]$ on its domain an adversary must pad the input itself before giving it to the oracle. Thus queries must be at least n bits long and the number of queries made by an adversary is upper bounded by μ/n where μ is the total length of messages queried by the adversary.

For the remainder of the paper we let $f : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ be an arbitrary, fixed function family. We consider H as a function family of signature $\{0,1\}^{4\kappa} \times \mathrm{Dom}(H) \to \{0,1\}^n$, where $H_{k_1 k_2 k_3 k_4}(x) := H[f_{k_1}, f_{k_2}, f_{k_3}, f_{k_4}](x)$. Likewise we consider F as a function family of signature $\{0,1\}^{3\kappa} \times \{0,1\}^{2n} \to \{0,1\}^n$ defined by $F_{k_1 k_2 k_3}(x \| y) = F[f_{k_1}, f_{k_2}, f_{k_3}](x \| y)$.

4 Security of SS-NMAC as a MAC

4.1 Overview

In this section we outline the proof that SS-NMAC is a secure MAC when $f_1, \ldots, f_4$ are secure MACs. The proof shows that H is a secure MAC family if f is a secure MAC family. In fact,

$$\mathbf{InSec}^{\mathrm{mac}}_H(t, \tilde{q}, \mu) \leq \left(1 + 30q^2 \log^2(q)\right) \cdot \mathbf{InSec}^{\mathrm{mac}}_f(t + O(q^2 n), q, qn) \tag{1}$$

where $q = \mu/n$ ($\tilde{q}$ is inconsequent, though one automatically has $\tilde{q} \leq q$). The $O(q^2 n)$ difference in running time is due to the overhead of a simulator.

Like Dodis, Pietrzak and Puniya [8], our security proof follows the approach developed by An and Bellare [2], who reduce the VIL-MAC security to FIL-WCR security. In order to summarize their method in a convenient way we refer to the members of a function family as being MAC-secure or WCR-secure (see section 2 for the definition of WCR security) though security is really a property of the function family. An and Bellare reduce the MAC security of a VIL function to the WCR security of a FIL function in two steps:

Step 1: The composition of a secure FIL-MAC f_k and a secure WCR function $G_{k'}$ is a secure VIL-MAC $f_k(G_{k'}(\cdot))$ (Lemma 4.2 [2]). Applying this to the case where $f_k = f_{k_4}$ and $G_{k'} = G[f_{k_1}, f_{k_2}, f_{k_3}]$ it therefore suffices to show that $G[f_{k_1}, f_{k_2}, f_{k_3}]$ is WCR-secure if f is a secure MAC family in order to show that $H[f_{k_1}, f_{k_2}, f_{k_3}, f_{k_4}] = f_{k_4}(G[f_{k_1}, f_{k_2}, f_{k_3}])$ is a secure MAC family.

Step 2: On a suffix-free domain of inputs the Merkle-Damgard iteration of a FIL-WCR compression function gives a VIL-WCR function (Lemma 4.3 [2]). Thus, by step 1, it suffices to show that the Shrimpton-Stam compression function $F[f_{k_1}, f_{k_2}, f_{k_3}]$ is FIL-WCR when f is a secure MAC family.

Steps 1 and 2 give a qualitative description of An and Bellare's approach. Quantitatively, their Lemmas 4.2 and 4.3 imply that

$$\mathbf{InSec}^{\mathrm{mac}}_H(t, \tilde{q}, \mu) \leq \mathbf{InSec}^{\mathrm{mac}}_f(t, q, qn) + \mathbf{InSec}^{\mathrm{wcr}}_F(t, q, 2qn) \tag{2}$$

where $q = \mu/n$. Since $\mathbf{InSec}^{\mathrm{mac}}_f(t, q, qn) \leq \mathbf{InSec}^{\mathrm{mac}}_f(t + O(q^2 n), q, qn)$ it therefore suffices to prove

$$\mathbf{InSec}^{\mathrm{wcr}}_F(t, q, 2qn) \leq 30q^2 \log^2(q) \cdot \mathbf{InSec}^{\mathrm{mac}}_f(t + O(q^2 n), q, qn) \tag{3}$$

in order to prove (1). Inequality (3) is really the paper's main result, and we state it as a theorem:

Theorem 1 *Let* $f : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ *and let* $F : \{0,1\}^{3\kappa} \times \{0,1\}^{2n} \to \{0,1\}^n$ *given by* $F_{k_1k_2k_3}(x\|y) = F[f_{k_1}, f_{k_2}, f_{k_3}](x\|y) = f_{k_1}(x) \oplus f_{k_3}(f_{k_1}(x) \oplus f_{k_2}(y))$. *Then*

$$\mathbf{InSec}^{\mathrm{wcr}}_F(t, q, 2qn) \leq 30q^2 \log^2(q) \cdot \mathbf{InSec}^{\mathrm{mac}}_f(t + O(q^2 n), q, qn).$$

The full proof of Theorem 1 is in Appendix B, but we give an outline in the next section.

Together with Lemmas 4.2 and 4.3 of [2], Theorem 1 implies inequality (3), which we restate as our theorem characterizing the MAC security of SS-NMAC:

Theorem 2 *Let* $f : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ *and let* $H : \{0,1\}^{4\kappa} \times \mathrm{Dom}(H) \to \{0,1\}^n$ *be the SS-NMAC function family. Then, letting* $q = \mu/n$,

$$\mathbf{InSec}^{\mathrm{mac}}_H(t, \tilde{q}, \mu) \leq \left(1 + 30q^2 \log^2(q)\right) \cdot \mathbf{InSec}^{\mathrm{mac}}_f(t + O(q^2 n), q, qn).$$

4.2 Proof Outline

In this section we give a proof of Theorem 1 under several simplifying assumptions, which make our presentation considerably easier, while maintaining the key ideas of the full proof. Recall, we need to upper bound the WCR-insecurity of the Shrimpton-Stam compression function F in terms of the MAC-insecurity of f. Equivalently, we must lower bound the MAC-insecurity of f in terms of the WCR-insecurity of F. To do the latter, we show how an ε-collision-finding adversary A for F can be turned into a δ-MAC-forging adversary B for f, where B uses the same number of queries but has chance of success $\delta = \Omega(\varepsilon/q^2 \log^2(q))$.

First, instead of giving A oracle access to $F[f_1, f_2, f_3]$, we directly give it oracle access to f_1, f_2, f_3, with q queries allowed to each f_i. Clearly, such an adversary can simulate q queries to F, so we only made A more powerful. (Note, this strengthened attacker will be useful when we extend our argument to the "oracle cipher" model in Section 6.) Let us generally denote the inputs to f_1, f_2, f_3 by x, y, z, respectively, and also denote by $x_1 \ldots x_q$, $y_1 \ldots y_q$ and $z_1 \ldots z_q$ the ordered inputs to f_1, f_2, f_3 supplied by A. As expected, the forger B will simulate this adversary A when trying to forge one of the f_i's, by using its own oracle to simulate the corresponding f_i, and simulating the other f_j's by picking their secret keys by itself and answering honestly.

Simplifying Assumptions. Before proceeding further, we state our simplifying assumptions on the behavior of A, which will make our construction of B much simpler, while retaining the key ingredients of the general case.

- **(No Collision in f_i's)** For any distinct inputs x_r and x_s that $\mathcal{A}$ supplied to f_1, $f_1(x_r) \neq f_1(x_s)$. Similar conditions also hold for f_2 and f_3.
- **(Query Order)** All the calls to f_1 and f_2 are made by A before any call to f_3 is made.

Let us briefly comment on these assumptions. The first assumption regarding the collisions in the f_i's is very minor, and is done for convenience only. Indeed, in the actual applications, the f_i's are *permutations*, so the assumption is trivially true. And even if the f_i's are arbitrary length-preserving MACs, the failure to satisfy our assumption with probability $\Omega(\varepsilon)$ trivially leads to a simple attacker B, forging the corresponding f_i with probability $\Omega(\varepsilon/q^2)$, by simply guessing the indices $r, s \in \{1 \ldots q\}$ of the colliding queries. So the only "real" assumption we make is the **Query Order** Assumption. This assumption is provably impossible for the "initial" attacker who has oracle access to $F[f_1, f_2, f_3]$, as opposed to f_1, f_2, f_3 (since f_3 will be called on the first call there), and is even more unreasonable for the generalized attacker that can query the f_i's in any order it wants. However the assumption is used

in a rather weak way in the proof sketch, as we will see, so that eliminating it only requires additional casework, and no significant new ideas.

NOTATION AND TERMINOLOGY. A *ball* is a pair (x, y) where $x, y \in \{0,1\}^n$. A *bin* is a value $z \in \{0,1\}^n$. It is instructive to associate balls (x, y) with the inputs to F, and the bins z with the inputs to f_3. After A makes q queries $x_1 \dots x_q$ to f_1 and $y_1 \dots y_q$ to f_2, we get $Q = q^2$ potential balls (x_r, y_s) "thrown" by A. In particular, we will say that such (x_r, y_s) is *placed* into the bin $z = f_1(x_r) \oplus f_2(y_s)$, and let $\mathsf{Bin}(z) = \{(x_r, y_s) : f_1(x_r) \oplus f_2(y_s) = z\}$ denote the set of balls placed into bin z. Notice, each query x_r to f_1 allows the attacker to simultaneously place $j \le q$ balls $(x_r, y_1), \dots, (x_r, y_j)$, where j is the number of queries to f_2 made so far. However, under our **No Collision** assumption of f_2, all these j balls go to *distinct* bins $f_1(x_r) \oplus f_2(y_s)$, where $1 \le s \le j$. Similar discussion holds for the calls to f_2. Also, under our **Query Order** Assumption, the attacker A places all Q balls into the appropriate bins in the first stage, before making any of its queries $z_1 \dots z_q$ to f_3 in the second stage. And after each such query z_t to f_3, A learns the value of $F(x\|y) = f_1(x) \oplus f_3(z_t)$ *precisely for all* $(x, y) \in \mathsf{Bin}(z_t)$.

BACK TO REDUCTION. By our assumption, A will find a collision $(x, y) \neq (x', y')$ to F with probability ε. Without loss of generality, we assume that A makes the queries necessary to verify this collision. Thus, $x, x' \in \{x_1, \dots, x_q\}$, $y, y' \in \{y_1, \dots, y_q\}$, and $z, z' \in \{z_1, \dots, z_q\}$, where $z = f_1(x) \oplus f_2(y)$ and $z' = f_1(x') \oplus f_2(y')$. Notice, under our **No Collision** assumption on f_1 and f_2, we claim that $z \neq z'$. Otherwise, $f_3(z) = f_3(z')$ and $f_1(x) \oplus f_3(z) = F(x\|y) = F(x'\|y') = f_1(x') \oplus f_3(z')$ imply that $f_1(x) = f_1(x')$, meaning that $x = x'$. Then $f_2(y) = f_1(x) \oplus z = f_1(x') \oplus z' = f_2(y')$, so $y = y'$, meaning that $(x, y) = (x', y')$. Hence, the "colliding" bins z and z' queried by A must be *distinct*.

We now define a key parameter which will determine the behavior of our forger B: *the maximum bin size* $m = \max_z |\mathsf{Bin}(z)|$ after the calls to f_1 and f_2 (the "filling" stage). We consider two complementary cases: (1) A finds a collision and $m \le \log(q)$, meaning that every bin z contains at most $\log(q)$ balls after the calls to f_1 and f_2 are finished; and (2) $m > \log(q)$, meaning that A managed to produce more than $\log(q)$ pairs (x_r, y_s) resulting in the same value $z = f_1(x_r) \oplus f_2(y_s)$.

(Interestingly, this parameter m corresponds to the largest "multi-collision" generated by A in the "filling" stage. As argued by Shrimpton and Stam [19] for the case of *truly random* functions f_i and $q \approx 2^{n/2}$, the value m must be smaller than $n^{1+o(1)} \approx \log q$ with high probability, more or less corresponding to saying that the attacker A must almost always be in case (1).)

By assumption that A succeeds to find a collision with probability $\ge \varepsilon$, at least one of these complementary cases happens with probability $\ge \varepsilon/2$.

Case (1): A finds a collision and $m \le \log(q)$. This is the "easy" case. Intuitively, by querying at most q bins z in the second stage, A learned the value of F in at most $qm \le q\log(q)$ points (x, y). As we will see, it will allow B to guess the colliding points $(x, y), (x', y')$ with probability $1/(q\log(q))^2$, and then forge the value $f_3(z') = f_3(z) \oplus f_1(x) \oplus f_1(x')$. More formally, B starts by choosing two random indices $j < i$ between 1 and q. Let z_i, z_j be the i-th and j-th queries made to f_3. When the query $f_3(z_i)$ is made, B chooses random elements $(x_i, y_i) \in \mathsf{Bin}(z_i)$ and $(x_j, y_j) \in \mathsf{Bin}(z_j)$ (assuming these sets are nonempty, otherwise B gives up), and predicts that $f_3(z_i) = f_1(x_i) \oplus f_1(x_j) \oplus f_3(z_j)$. Notice, this corresponds to guessing that $F(x_i, y_i) = F(x_j, y_j)$, which implies that A is about to find a collision. This strategy cannot be successful unless A finds a collision (which we are assuming happens in this case), and unless the colliding bins z_i and z_j are distinct, which we also argued earlier as following from our **No Collision** assumption. But when A does find a collision, B's chance of guessing the indices i, j correctly is $1/\binom{q}{2} \ge 1/q^2$. Moreover if $\max_z |\mathsf{Bin}(z)| \le \log(q)$, B's chance of guessing the right elements (x_i, y_i) and (x_j, y_j) in $\mathsf{Bin}(z_i)$ and $\mathsf{Bin}(z_j)$ is at least $1/\log(q)$ each. Thus B's chance of success with this strategy is at least $1/q^2\log^2(q)$ when $\max_z |\mathsf{Bin}(z)| \le \log(q)$ and A finds a collision.

Case (2): A produces $m > \log(q)$. This is the "hard" case, where our balls-and-bins terminology comes in handy. Intuitively, if A throws q^2 balls with a guarantee that some bin will contain a lot of

balls at the end, B should have a non-trivial chance (analyzed below) to guess the bin z corresponding to some ball (x, y) *before this ball is thrown.* To effect such a guess, when A "throws" the ball (x, y) by querying, say, $f_1(x)$ after previously querying $f_2(y)$, B can predict that $f_1(x) = z \oplus f_2(y)$, or conversely with f_1 and f_2 reversed if A queries $f_2(y)$ after querying $f_1(x)$. In other words, predicting the output of f_1 or f_2 on a value queried by A is *equivalent* to predicting the bin where a particular ball (x, y) will land at the point when the latest of the two queries $f_1(x)$, $f_2(y)$ is made. Thus, we may view B's task as consisting of observing a set of $Q = q^2$ balls being placed by groups in 2^n bins, and interrupting the game at some point to *predict the bin where a particular ball that is about to be placed.* We model this by a "balls-and-bins" game played by A and B, where A is incrementally throwing Q balls into bins, trying to fill some bin with more than $\log(q)$ balls, and yet without having B be able to guess the position of a ball before it is placed. Based on our discussion, the precise "rules" of this game are as follows:

Balls-and-Bins Game:

- The game proceeds in $2q$ rounds, after which A is required to throw exactly $Q = q^2$ balls.
- Before each round, A announces to B at most q balls $b_1, \ldots, b_t$ that it will be throwing into (necessarily) *distinct* bins in this round. [Intuitively, a round corresponds to a query to $f_1(x_r)$ (or $f_2(y_s)$), and the balls are the corresponding values (x_r, y_j) (or (x_i, y_s)) for prior x_i's or y_j's.]
- In turn, B can secretly "pass" or make a "guess" (ℓ, z) that the ball b_ℓ will be thrown into bin z (where $1 \leq \ell \leq t$). [Intuitively, a successful guess will allow B to forge either f_1 or f_2, as outlined earlier.]
- A announces to B the bins where $b_1 \ldots b_t$ are thrown. [Intuitively, B learns the value of f_1 or f_2 at the queried point, allowing it to learn the bin identities.]
- If B made a guess during this round, B wins the game if the guess is correct, and loses otherwise. If B did not make a guess, proceed to the next round.
- B must make a guess at some round, while A must fill at least one bin with more than $(\log q)$ balls.

Lemma 1 *Irrespective of A's strategy, there exists an efficient strategy for B to win the above game with probability at least $1/4q^2$ whenever some bin contains more than $\log(q)$ balls at the end of the game.*

Proof. B's strategy is relatively simple:

1. Choose a random index i between 1 and q^2, and a second random integer k between 1 and $\log(q)$.
2. Pass in all the rounds before the i-th overall ball is about to be thrown.
3. When the i-th ball is about to be thrown, make a secret guess that this ball will be thrown in a random bin z chosen among those bins *already containing at least k balls prior to this round* (or guess any bin if no such bin exists).

We argue that with this strategy, B's chance of success is at least $1/4q^2$, provided that some bin contains more than $\log(q)$ balls by the end of the game. Let c_j be the total number of balls that are thrown into bins that already have at least j balls in them right before the round when this ball is thrown. Thus $c_0 = q^2$ and $c_{\log(q)} \geq 1$ by assumption that a "heavy" bin exists at the end of the game. Also note that c_j is an upper bound for the number of bins that have $j + 1$ balls in them at the end of the game, since for a bin to receive $j + 1$ balls, some ball has to be thrown into it when the bin already has j balls.

For a fixed value of k, B's chance of correctly guessing the bin is at least $\frac{c_k}{q^2} \cdot \frac{1}{c_{k-1}} = \frac{1}{q^2} \cdot \frac{c_k}{c_{k-1}}$. This is because B has chance at least $\frac{c_k}{q^2}$ of choosing a ball thrown into a bin with at least k balls, and then has at least chance $\frac{1}{c_{k-1}}$ of choosing the bin correctly, given that there are at most c_{k-1} bins with k balls in them even at the end of the game, let alone in some intermediate round. Summing over the different values of k (which each have chance $1/\log(q)$ of being selected), we thus see that B's chance of success is

$$\begin{aligned}
\sum_{k=1}^{\log(q)} \frac{1}{\log(q)} \cdot \frac{1}{q^2} \cdot \frac{c_k}{c_{k-1}} &= \frac{1}{q^2}\ \mathsf{ArithmeticMean}\left(\frac{c_1}{c_0}, \ldots, \frac{c_{\log(q)}}{c_{\log(q)-1}}\right) \\
&\geq \frac{1}{q^2}\ \mathsf{GeometricMean}\left(\frac{c_1}{c_0}, \ldots, \frac{c_{\log(q)}}{c_{\log(q)-1}}\right) \\
&= \frac{1}{q^2}\left(\frac{c_{\log(q)}}{c_0}\right)^{\frac{1}{\log(q)}} \geq \frac{1}{q^2}\left(\frac{1}{q^2}\right)^{\frac{1}{\log(q)}} \\
&= \frac{1}{4q^2}
\end{aligned}$$

as claimed, where we used $c_0 = q^2$ and $c_{\log(q)} \geq 1$. □

The above lemma immediately gives us a forger B for Case (2), which succeeds to forge either f_1 or f_2 with probability at least $1/4q^2 > 1/(q\log(q))^2$ for that case. As B chooses randomly which strategy to use and one of the two cases must occur with probability at least $\varepsilon/2$, B's chance of success is at least $\frac{\varepsilon}{2}\min(1/(q\log(q))^2, 1/4q^2) = \varepsilon/2(q\log(q))^2$, completing the WCR proof under our two simplifying assumptions.

General Case. Note that our (main) **Query Order** Assumption (namely that queries to f_3 come before queries to f_1 and f_2) is only used rather weakly, in the sense that A could make its queries in any order as long as the query which completes the collision is a query to f_3. Thus removing this assumption amounts to handling two extra cases, in which collisions are completed with queries to f_1 or f_2 instead of f_3. It turns out these cases can be handled fairly similarly to the f_3 case. The details are in Appendix B.

5 Security of SS-NMAC as a PRF

In this section we show that SS-NMAC is a secure PRF if f is a secure PRF. We will prove a stronger property in Section 6; here we give a proof reducing to the security of encrypted CBC-MAC, which gives a weaker result but a better security bound. The precise statement is the following theorem.

Theorem 3 *Let $f : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ and let H be the SS-NMAC function family. Then, letting $q = \mu/n$ and $\varepsilon = \mathbf{InSec}^{\mathrm{prf}}_f(t, q, qn)$, we have*

$$\mathbf{InSec}^{\mathrm{prf}}_H(t, \tilde{q}, \mu) \leq 5q^2/2^n + 4\varepsilon.$$

Proof. Let H^* be the SS-NMAC construction where f_1, f_2, f_3, f_4 are random functions. Then obviously $\mathbf{InSec}^{\mathrm{prf}}_H(t, \tilde{q}, \mu) \leq \mathbf{InSec}^{\mathrm{prf}}_{H^*}(t, \tilde{q}, \mu) + 4\varepsilon$, so it suffices to show that $\mathbf{InSec}^{\mathrm{prf}}_{H^*}(t, \tilde{q}, \mu) \leq 5q^2/2^n$ where $q = \mu/n$.

We show that $\mathbf{InSec}^{\mathrm{prf}}_{H^*}(t, \tilde{q}, \mu) \leq 5q^2/2^n$ by reducing to the security of the "original" encrypted CBC-MAC, which is defined using a function family f of n-bit to n-bit functions by

$$C[f_1, f_2](x_1 \| \ldots \| x_m) = f_2(\ldots f_1(f_1(x_1) \oplus x_2) \ldots)$$

Let C^* be the instance of C where f_1, f_2 are random functions (namely, f is the set of all functions $\{0,1\}^n \to \{0,1\}^n$). It is known that $\mathbf{InSec}^{\mathrm{prf}}_{C^*}(t,\tilde{q},\mu) \leq q^2/2^n$ where still $q = \mu/n$ [15]. The security proof in [15] is also easily seen to apply to the case of a three-keyed, "alternating" encrypted CBC-MAC defined by

$$C_A[f_2,f_3,f_4](x_1\|\dots\|x_m) = f_4(\dots f_2(f_3(f_2(x_1)\oplus x_2)\oplus x_3)\dots)$$

in which encryptions by f_2 and f_3 alternate. Thus $\mathbf{InSec}^{\mathrm{prf}}_{C^*_A}(t,\tilde{q},\mu) \leq q^2/2^n$ where C^*_A is the random function implementation of C_A.

Note that C_A becomes H if each block of input is repeated once and encrypted with a call to f_1. Thus a distinguisher D for H^* can be used to obtain a distinguisher D' for C^*_A: sample a key k_1 to simulate the function f_{k_1}, then simulate a query $x_1\|\cdots\|x_m$ of D to the oracle H^* by passing $f_{k_1}(x_1)\|f_{k_1}(x_1)\|f_{k_1}(x_2)\|f_{k_1}(x_2)\cdots\ f_{k_1}(x_m)\|f_{k_1}(x_m)$ to the oracle for C^*_A.

If the oracle is a true instance of C^*_A the answers returned to D look exactly as the answers of an oracle to H^*, so D's chance of distinguishing correctly is unaffected in that case. If on the other hand the oracle is a random function the answers returned to D are independent random values except when the same input is queried twice to the random oracle, which can happen because of collisions in f_{k_1}. The chance of a collision in f_{k_1} when $q = \mu/n$ blocks of message are queried and f_{k_1} is a random function is at most $q^2/2^n$, however, so D and D''s distinguishing advantages differ by at most $q^2/2^n$. Thus, since D' uses twice the message length, we get the desired

$$\mathbf{InSec}^{\mathrm{prf}}_{H^*}(t,\tilde{q},\mu) \leq \mathbf{InSec}^{\mathrm{prf}}_{C^*_A}(t,\tilde{q},2\mu) + q^2/2^n \leq 5q^2/2^n.$$

□

6 Enhanced PRF Security in the Oracle Cipher Model

In this section, we introduce (following [9]) a strictly *stronger* PRF security notion for block-cipher-based PRFs in the so called *oracle cipher model*, and show that SS-NMAC has (nearly) birthday "oracle cipher security" when instantiated with a secure PRP.

Let H be a function using a fixed-key block cipher f (or a small set of different fixed key block ciphers). Essentially, the oracle cipher model is designed to allow the adversary to view computation transcripts of H, but not including the internals of the block cipher calls. For example, one can imagine that the adversary witnesses a trusted party's computation of H on various inputs, where the trusted party out-sources the block cipher calls to a smart-card, so that the secret keys remain hidden from the adversary. We argue that H is a good random function if, subsequent to viewing a number of such computations, the adversary is unable to distinguish H (queried on new values) from a truly random function.

Let M^f be an oracle Turing machine implementing H. Before the game starts random keys are chosen for the block ciphers, a random function h with same domain and range as H is sampled, and a coin flipped to determine whether the adversary will be in the "real world" or "random world". We allow the adversary two types of queries: "transcript" queries and "oracle" queries. When the adversary A makes a transcript query the transcript of the computation $M^f(x)$ is returned to A. When the adversary makes a oracle query (oracle queries must be distinct from transcript queries), the adversary either gets $H^f(x)$ or $h(x)$ depending on whether it is in the real world or the random world. The adversary wins if it can distinguish the two worlds.

We call the advantage of an adversary at winning this game the *oracle cipher PRF security of* H, denoted $\varepsilon_{\mathrm{oprf}}$. Clearly $\varepsilon_{\mathrm{oprf}} \geq \varepsilon_{\mathrm{prf}}$ for the same number of queries and the same computational resources, since the adversary is free to play the oracle cipher game without making any transcript queries. Let $\mathbf{InSec}^{\mathrm{oprf}}_{H}(t,\tilde{q},\mu)$ be the maximum $\varepsilon_{\mathrm{oprf}}$ over all adversaries running in time at most t, making at most q queries of total (padded) length at most μ, where the running time includes the time necessary to run H

and M^f. (Obviously, $\mathbf{InSec}^{\mathrm{oprf}}_H(t, \tilde{q}, \mu)$ implicitly depends on the choice of M.) We have the following theorem showing that the oracle cipher security of SS-NMAC is nearly equivalent to its standard PRF security.

Theorem 4 *Let $f : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$, let $H : \{0,1\}^{4\kappa} \times \mathrm{Dom}(H) \to \{0,1\}^n$ be the SS-NMAC function family, and let M^{f_1,f_2,f_3,f_4} be the natural oracle Turing implementation of SS-NMAC , which makes $3\ell+1$ oracle calls to compute $H(x)$ on a padded input of ℓ blocks. Then with respect to this oracle Turing machine, and letting $q = \mu/n$ and $\varepsilon = \mathbf{InSec}^{\mathrm{prf}}_f(t, q, qn)$, we have*

$$\mathbf{InSec}^{\mathrm{oprf}}_H(t, \tilde{q}, \mu) \leq 30q^2 \log^2(q)/2^n + 4\varepsilon.$$

Proof. Let H^* be the instantiation of H with a truly random function family instead of with f. We clearly have $\mathbf{InSec}^{\mathrm{oprf}}_H(t, \tilde{q}, \mu) \leq \mathbf{InSec}^{\mathrm{oprf}}_{H^*} + 4\varepsilon$, so it suffices to show $\mathbf{InSec}^{\mathrm{oprf}}_{H^*} \leq 30q^2 \log^2(q)/2^n$.

We now modify the game like so: for each type of query (transcript and oracle), the adversary is allowed to view the transcript of the computation of $H^*(x)$ *up to the application of* f_4. Then for a transcript query the actual application of f_4 is shown as part of the transcript to the adversary, whereas for an oracle query the value of the oracle query is simply appended to the transcript (which will be the value of f_4 in the real world, or else simply the value of the random function h). Note the adversary knows in either case which type of query it is witnessing, but cannot independently verify f_4 for oracle queries unless it happens to make another query later (either transcript or oracle) which results in the same input to f_4. In fact, if the adversary never makes two queries at least one of which is an oracle query that result in the same input to f_4, the two worlds look exactly alike (because f_4 is uniformly random) and the adversary has zero advantage.

Thus the adversary's advantage is upper bounded by its probability of finding a collision at the input to f_4 with free oracle access to f_1, f_2, f_3, which is in turn upper bounded by the collision resistance of the Shrimpton-Stam compression function when instantiated with random functions. Thus Theorem 1 applied with MAC insecurity $1/2^n$ gives $\mathbf{InSec}^{\mathrm{oprf}}_{H^*}(t, \tilde{q}, \mu) \leq 30q^2 \log^2(q)/2^n$, as desired. □

7 Unpredictability vs. Pseudorandomness

Given that our solution is three times slower than CBC-MAC, it is interesting to see if existing block ciphers, such as AES, are indeed more unpredictable than pseudorandom. Notice, even if our n-bit block cipher is completely ideal, it has security $\varepsilon_{\mathrm{prf}} \sim q^2/2^{n+1}$ as a one-block PRF, and a much better security $\varepsilon_{\mathrm{mac}} \sim 1/(2^n - q)$ as a one-block MAC, where q is the number of input queries issued by the attacker. Also, in theory is is trivial to construct artificial block ciphers which are much more unpredictable than pseudorandom. Unfortunately, existing block ciphers are neither ideal nor artificial. For such "real" block ciphers, to the best of our knowledge, this gap between unpredictability and pseudorandomness has not been researched extensively. In part, this might be due to the cryptanalytic "culture" to call the attack truly "successful" if it actually recovers the secret key, which, obviously, will not demonstrate the gap we are seeking here.

We give a (rather weak) example to demonstrate this point. It is well known in complexity theory [22] that no pseudorandom generator with κ-bit key can have security more than $2^{-\kappa/2}$ (against non-uniform attackers), even against *linear tests*.[3] This means that no non-trivial PRF with a κ-bit key can have security $\varepsilon_{\mathrm{prf}} \leq 2^{-\kappa/2}$, even for $q = O(1)$ (e.g., *AES cannot be more than* 2^{-64} *secure, even for* $q = 2$*!*). In contrast, no such limitation is known for unpredictability, even for exponentially high number of queries q (e.g., for all we know, AES might be almost 2^{-128} secure, even for $q = 2^{60}$ or higher). However, the

[3] Since an ε-secure pseudorandom generator must also be an ε-biased set [13], and such sets must have seed length at least $2\log(1/\varepsilon)$ (see [1]). Thus, $\kappa < 2\log(1/\varepsilon)$.

above theoretical "separation" is not considered a "real attack", since the best known way to translate this specific $2^{-\kappa/2}$ distinguishing attack to the key recovery attack takes time $\Omega(2^{\kappa})$, which is trivial.

We hope that our work will motivate further research to understand the gap between unpredictability and pseudorandomness of existing block ciphers, such as AES. In particular, to answer the question if existing modes, such as CBC-MAC or HMAC, should be replaced by slower, but more "resilient" modes, such as SS-NMAC.

References

[1] Noga Alon, Oded Goldreich, Johan Hastad, Rene Peralta, *Simple Construction of Almost k-wise Independent Random Variables*. Random Struct. Algorithms, 3(3):289–304, 1992.

[2] Jee Hea An, Mihir Bellare, *Constructing VIL-MACs from FIL-MACs: Message Authentication under Weakened Assumptions*, CRYPTO 1999, pages 252–269.

[3] Mihir Bellare, *New Proofs for NMAC and HMAC: Security without Collision-Resistance*, CRYPTO 2006, pages 602–619.

[4] Mihir Bellare, Joe Kilian, Phillip Rogaway, *The Security of Cipher Block Chaining*, CRYPTO 1994, pages 341–358.

[5] Mihir Bellare, Ran Canetti, Hugo Krawczyk, *Pseudorandom Functions Re-visited: The Cascade Construction and Its Concrete Security*, FOCS 1996, pages 514–523.

[6] Mihir Bellare, Ran Canetti, Hugo Krawczyk, *Keying Hash Functions for Message Authentication*, CRYPTO 1996, pages 1–15.

[7] John Black, Phillip Rogaway, Thomas Shrimpton, *Black-Box Analysis of the Block Cipher-Based Hash-Function Constructions from PGV*, CRYPTO 2002, pages 320–335.

[8] Yevgeniy Dodis, Krzysztof Pietrzak, Prashant Puniya, *A New Mode of Operation for Block Ciphers and Length-Preserving MACs*, EUROCRYPT 2008, pages 198–219.

[9] Yevgeniy Dodis, Prashant Puniya, *Feistel Networks Made Public, and Applications*, EUROCRYPT 2007, pages 534–554.

[10] Yevgeniy Dodis, John Steinberger, *Message Authentication Codes from Unpredictable Block Ciphers*. Full version of this paper. Available at `http://people.csail.mit.edu/dodis/ps/tight-mac.ps`.

[11] Oded Goldreich and Leonid Levin, *A hard-core predicate for all one-way functions*, STOC 1989, pages 25–32.

[12] Michael Luby and Charles Rackoff, *How to construct pseudo-random permutations from pseudo-random functions*, SIAM J. Comput. 17(2):373–386, 1988.

[13] Joseph Naor, Moni Naor. *Small-Bias Probability Spaces: Efficient Constructions and Applications*. SIAM J. Comput. 22(4):838–856, 1993.

[14] Momi Naor, Omer Reingold, *From unpredictability to indistinguishability: A simple construction of pseudo-random functions from MACs*, CRYPTO 1998, pages 267–282.

[15] Erez Petrank, Charles Rackoff, *CBC MAC for Real-Time Data Sources*, J. Cryptology 13(3):315–338, 2000.

[16] Bart Preneel, Rene Govaerts, Joos Vandewalle, *Hash Functions Based on Block Ciphers: A Synthetic Approach*, CRYPTO 1993, pages 368-378.

[17] Bart Preneel and Paul C. van Oorschot, *MD-x MAC and building fast MACs from hash functions*, CRYPTO 1995, pages 1–14.

[18] Phillip Rogaway and John Steinberger, *How to Build a Permutation-Based Hash Function*, CRYPTO 2008, pages 433–450.

[19] Thomas Shrimpton and Martijn Stam, *Building a Collision-Resistant Compression Function from Non-Compressing Primitives*, ICALP 2008, pages 643–654. Also available at Cryptology ePrint Archive: Report 2007/409.

[20] Daniel R. Simon, *Finding Collisions on a One-Way Street: Can Secure Hash Functions Be Based on General Assumptions?*, EUROCRYPT 1998, pages 334–345.

[21] Xiaoyun Wang, Hongbo Yu, *How to break MD5 and Other Hash Functions*, EUROCRYPT 2005, pages 19–35.

[22] David Zuckerman, *Private communication.*

A Inapplicability of Existing Solutions

Here survey some of the existing approaches, and explain why they do not appear to give a satisfactory answer to our main question.

REDUCING TO PRF. Given a large number of proposals which are secure when the block cipher is pseudorandom, such as (encrypted) CBC-MAC [4,15] and HMAC/NMAC [3,6], the most natural approach is to first build a (fixed-input-length) PRF out of our block cipher, and then apply the existing machinery. Unfortunately, such generic conversions from unpredictability to pseudorandomness, originating with the revolutionary paper of Goldreich and Levin [11], usually achieve pretty poor parameters. In our setting, the most efficient PRF built from an unpredictable function (or permutation) is the construction of Naor and Reingold [14], who showed that the Goldreich-Levin construction *with a secret inner-product vector r* yields a 1-bit PRF with PRF-security $\varepsilon_{\text{prf}} \approx (\varepsilon_{\text{up}} \cdot q_{\text{prf}}^2)^{1/2}$, where q_{prf} is the number of queries the adversary makes to the PRF and ε_{up} is the security of the block cipher under $q = O(q_{\text{prf}}^3/\varepsilon_{\text{prf}}^2)$ queries. Since $q \leq 2^n$, the bound is void when $q_{\text{prf}} \geq 2^{n/3}$ or $\varepsilon_{\text{prf}} \leq \frac{1}{2^{n/2}}$, and (in particular) birthday security cannot be achieved. Moreover, to use the resulting 1-bit PRF, we first need to amplify its output length to be at least the security parameter κ, resulting in bad efficiency rate.

Thus, not surprisingly, the route through pseudorandomness is too inefficient. Hence, we turn to other "direct" and efficient VIL-MAC constructions to see if, by "lucky chance", they happen to be secure with unpredictable block ciphers.

CBC MODE. The popular CBC-MAC (and it related variants) is a simple, rate-1 construction of a VIL-MAC from a block cipher. In fact, under the assumption that the block cipher is a PRP, one can even get a VIL-PRF with birthday security of $2^{n/2}$ messages. Unfortunately, as demonstrated by An and Bellare [2], the CBC-MAC could be completely insecure assuming the block cipher is a UP: $\varepsilon_{\text{mac}} = 1$ already for two queries on two-block messages! Hence, this approach fails completely.

HMAC/NMAC. The popular HMAC construction [6] is a heuristic practical variant of the NMAC construction, and is based on the Merkle-Damgard (i.e., cascade) mode of operation. It requires a keyed function family whose output length is at least as large as its key length, which is the case for many block ciphers including AES. In each iteration of the cascade, the current chaining variable (initially, the key) is used as the key for the next iteration. Intuitively, this works for the case of PRFs/PRPs,

whose output is (pseudo)random, and can be used as a key. However, it totally fails for the case of UPs, whose output need not be random at all (just unpredictable). For example, it is very simple to construct (artificial) UPs, for which the cascading operation will converge to a known constant on some fixed two-block input.[4] Thus, simple counter-examples of this kind easily dispatch this approach.[5]

HASH-THEN-MAC. A classical approach to extend a domain of a cryptographic primitive is to use an appropriate variable-input-length hash function H to first hash a long message x to a short digest $y = H(x)$, and then to apply a given FIL-MAC (such as UP) to y. The key question is which properties of H suffice for this goal. The first simple observation is that collision-resistant hash function (CRHFs) obviously suffice. However, as explained in the Introduction, this is unsatisfactory both in theory and in practice.

Thus, we would like to base security of the "hash-then-mac" approach on weaker hash functions than CRHFs, which is promising since the hash function H can be *keyed*. In particular, (almost) *universal hash functions* are known to be sufficient for the soundness of this approach (this is attributed to [5] by [3]), when later composed with a FIL-PRF f (such as a PRP). Intuitively, all long as no collisions for H are produced so far, the "outer" f will hide all information about the key of H, allowing one to securely evaluate H on future inputs, without causing collisions with high probability. Unfortunately, it is very crucial that the outer f is pseudorandom, and it is easy to see that this approach will not work, in general, with *unpredictable* block ciphers.[6]

Luckily, as also explained in the Introduction, one can base the security of the "hash-then-mac" approach, even when the outer MAC function is only unpredictable, on a weaker class of hash functions than CRHFs, called *Weakly Collision-Resistant* (WCR) [6]. Moreover, a shrinking MAC is also a WCR with birthday security [2]. Coupled with a simple domain extension of WCR hash functions via the (strengthened) Merkle-Damgard transform [2], the question of building VIL-MACs from UPs is essentially equivalent to the question of building either a FIL-WCR hash or a sufficiently shrinking FIL-MAC from a UP. Unfortunately, as we show below, this latter question turns out to be quite non-trivial.

TRUNCATING BLOCK CIPHER? Of course, one can always turn a UP (i.e., a length-preserving MAC) into a shrinking MAC (or, perhaps, WCR hash), by simply truncating its output. Unfortunately, this does not work. First, truncating a bits can potentially degrade the unpredictability of the construction by a factor 2^a, meaning that this approach is theoretically sound only for very small a (logarithmic in security parameter). However, since the rate of the resulting construction is n/a, to achieve constant efficiency rate one must truncate at least a constant fraction of bits, making it unsound. Moreover, even if one somehow assumes that such "sufficiently truncated" block cipher is still a "good" MAC, the birthday security will now be with respect to the *truncated* output length $(n - a)$. E.g., truncating half the bits can give "birthday security" at most $2^{n/4}$, defeating our goal.

TWO-BLOCK "SOLUTION"? Given that practical MACs, such as CBC-MAC, are secure with PRPs, one can simply *assume away* the difficulty of the building a "sufficiently compressing" MAC (or WCR), and assume that two-block versions of these MACs yield a secure two-to-one MAC (or WCR). The "rationale" for this trivialization of the problem is the following. Perhaps, it is "almost as good" to assume that a two-block CBC is a secure MAC (or WCR) as to assume that a one-block version is a MAC (i.e., UP). And, maybe, this strange ad hoc assumption is somehow better than assuming a PRP, which provably implies it. Our belief is that this reasoning is extremely dangerous, and is really not substantiated by any evidence. In fact, we know that it is wrong with (generic) unpredictable block ciphers. Thus, we reject it.

[4] Say, $f_k(0^n)$ is unpredictable, but always begins with $0^{n/2}$, and $f_{0^{n/2}\|k'}(x) = x$ for all $k' \in \{0,1\}^{n/2}, x \in \{0,1\}^n$.

[5] Bellare [3] mentions some ad hoc assumptions on the components of NMAC/HMAC, under which one can prove (only) MAC security. As far as we can see, though, these relaxations (among others, so called "privacy-preserving MACs") seem to be only marginally weaker than assuming a PRP, and *much stronger* than assuming regular unforgeability, as done here.

[6] E.g., it is easy to construct a natural universal hash function and an artificial UP for which this approach is completely insecure.

FEISTEL NETWORK. Another popular approach to double the domain of a primitive is to use the Feistel Network, which iterates the permutation $\pi(x\|y) = (y\|f(y) \oplus x)$ for several rounds (with different keyed functions f in each round). The celebrated result of Luby and Rackoff [12] showed that already 3 rounds of Feistel would give a PRF from $2n$ to $2n$ bits, which could be truncated to n bits by dropping one of the halves. Unfortunately, this argument does not translate if we replace a PRP by a (general) UP. An and Bellare [2] showed that three rounds of Feistel fail to preserve unpredictability (or yield WCR), and Dodis and Puniya [9] extended this attack even to $O(\log \kappa)$ rounds, where κ is the security parameter. On a positive, [9] showed that super-logarithmic number $\omega(\log \kappa)$ of Feistel rounds indeed allows one to construct a two-to-one MAC (and WCR) from an unpredictable block cipher. Unfortunately, this results in super-constant rate $\omega(\log \kappa)$, making this construction somewhat inefficient for practice. More significantly, the security of this construction proven by [9] was only $\varepsilon_{\text{mac}} = O(\varepsilon_{\text{up}} \cdot q_{\text{mac}}^6)$, where q_{mac} is the number of queries made by the MAC-forging adversary, meaning that it can only be secure for at most $2^{n/6}$ messages, which is unacceptable for $n = 128$. To summarize, the work of [9] gives the first reasonable construction of a VIL-MAC from an UP, but has very poor security and super-constant rate.
ENHANCED CBC MODE. This mode was recently proposed by Dodis et al. [8]. The authors made the surprisingly simple observation that the function $h(x\|y) = f_1(x) \oplus f_2(y)$ is a two-to-one, rate-2 WCR hash function, assuming f_1 and f_2 are two independent UPs. This immediately gives a rate-2 VIL-MAC from UPs, which is very efficient, and is the first (and only) constant-rate solution to Question 1 known prior to this work. Unfortunately, the security of this WCR function is $O(\varepsilon_{\text{up}} \cdot q^4)$, since the UP forger has to guess all four queries $f_1(x), f_1(x'), f_2(y), f_2(y')$ which are "responsible" for the collision $h(x\|y) = h(x'\|y')$, and predict the last value in the quartet as the XOR of the three prior values. Moreover, it is easy to see that this bound is actually tight. Thus, the construction can only be secure for at most $2^{n/4}$ messages, again making it fall short of our goal of obtaining security up to $2^{n/2}$ asked in Question 2. In fact, this question of achieving "birthday security" $2^{n/2}$ was the main open question posed in [8], which we resolve in this work.

B Full Proof of Theorem 1

This section contains the proof of Theorem 1. As explained in the proof outline, A is a collision-finding adversary for F with oracle access to f_1, f_2, f_3 making q queries to each f_i; B simulates A, interrupting the computation to forge one of the f_i's at some un-queried point. We must exhibit a strategy for B such that B's chance of forging is at least $\varepsilon/10q^2\log^2(q)$ where ε is A's probability of finding a collision, and such that B's computational overhead is $O(q^2 n)$. We assume that A makes all the queries necessary to compute its collision.

At the start B chooses a random integer between 1 and 10, uniformly. Depending on this number B attempts one of 10 different types of forgeries. The types of forgeries are described below. All random choices are made uniformly in the specified domain.

Type 1: B chooses random indices $i < j$ between 1 and q. When A makes its j-th query x_j to f_1 B guesses the answer will be $f_1(x_i)$ where x_i was the i-th query to f_1.

Type 2: B chooses random indices $i < j$ between 1 and q. When A makes its j-th query y_j to f_2 B guesses the answer will be $f_2(y_i)$ where y_i was the i-th query to f_2.

Type 3: B chooses random indices $i < j$ between 1 and q. When A makes its j-th query z_j to f_3 B guesses the answer will be $f_3(z_i)$ where z_i was the i-th query to f_3.

Type 4: B chooses random indices $i < j$ between 1 and q. When A makes its j-th query z_j to f_3 B guesses the answer will be $z_j \oplus z_i \oplus f_3(z_i)$ where z_i was the i-th query to f_3.

Type 5: B chooses a random integer ℓ between 1 and q^2 and a random integer k between 1 and $\log(q)$. As A makes queries B keeps track of the pairs (x, y) for which A knows $f_1(x)$, $f_2(y)$ in a sequence S. Pairs are added to S as A makes queries: when A makes a query x to f_1, say, B adds the pairs $\{(x, y) : A$ has already queried $f_2(y)\}$ to S, with the pairs ordered lexicographically by y, and symmetrically when A makes queries f_2 on a new point y. The *value* of pair $(x, y) \in S$ is $f_1(x) \oplus f_2(x)$. When B sees that A's latest query will result in the length of S becoming $\geq \ell$ it finds the pair (x, y) that will be the ℓ-th pair of S (it can do this without seeing the answer to the query), chooses a value d uniformly in the set $c = \{c \in \{0,1\}^n :$ there are at least k pairs in S of value $c\}$, and guesses the answer of the query by guessing the value of (x, y) will be d (namely guesses $f_1(x) = d \oplus f_2(y)$ if A's query was $f_1(x)$ and guesses $f_2(y) = d \oplus f_1(x)$ if A's query was $f_2(y)$). No guessing occurs if C is empty.

Type 6: B chooses a random integer ℓ between 1 and q^2 and a random integer k between 1 and $\log(q)$. As A makes queries B keeps track of the pairs (x, z) for which A knows $f_1(x)$, $f_3(z)$ in a sequence T, which is constructed like the sequence S of the Type 5 strategy. Like for the sequence S, the *value* of pair $(x, z) \in T$ is $f_1(x) \oplus f_3(z)$. When B sees that A's latest query will result in the length of T becoming $\geq \ell$ it finds the pair (x, z) that will be the ℓ-th pair of T, chooses a value d uniformly in the set $C = \{c \in \{0,1\}^n :$ there are at least k pairs in T of value $c\}$, and guesses the answer of the query by guessing the value of (x, z) will be d. No guessing occurs if C is empty.

Type 7: B chooses a random integer ℓ between 1 and q^2 and a random integer k between 1 and $\log(q)$. As A makes queries B keeps track of the pairs (y, z) for which A knows $f_2(x)$, $f_3(z)$ in a sequence U, which is constructed like the sequences S, T of the Type 4 and Type 5 strategies. The *value* of pair $(y, z) \in U$ is $f_2(y) \oplus z \oplus f_3(z)$. When B sees that A's latest query will result in the length of U becoming $\geq \ell$ it finds the pair (y, z) that will be the ℓ-th pair of U, chooses a value d uniformly in the set $C = \{c \in \{0,1\}^n :$ there are at least k pairs in U of value $c\}$, and guesses the answer of the query by guessing the value of (y, z) will be d. No guessing occurs if C is empty.

Type 8: B chooses a random index i between 1 and q. When A makes its i-th query to f_1, say $f_1(x)$, B chooses one element at random from the set $Q = \{(y, z, x', y') : f_2(y) \oplus z \oplus f_3(z) = F(x'\|y')$ and A has made the queries $f_2(y)$, $f_3(z)$ and the queries necessary to compute $F(x'\|y')\}$ (if the set is nonempty) and guesses the answer to $f_1(x)$ will be $f_2(y) \oplus z$.

Type 9: B chooses a random index i between 1 and q. When A makes its i-th query to f_2, say $f_2(y)$, B chooses one element at random from the set $R = \{(x, z, x', y') : f_1(x) \oplus f_3(z) = F(x'\|y')$ and A has made the queries $f_1(x)$, $f_3(z)$ and the queries necessary to compute $F(x'\|y')\}$ (if the set is nonempty) and guesses the answer to $f_2(y)$ will be $f_1(x) \oplus z$.

Type 10: B chooses random indices $j < i$ between 1 and q. Let z_i, z_j be the i-th and j-th queries that A makes to f_3 and let $\mathsf{Bin}(z)$ be as defined in Section 4.2. When the query $f_3(z_i)$ is made, B chooses a random element $(x_i, y_i) \in \mathsf{Bin}(z_i)$ and a random element $(x_j, y_j) \in \mathsf{Bin}(z_j)$ (if these sets are nonempty, otherwise B gives up), and guesses that $f_3(z_i) = f_1(x_i) \oplus f_1(x_j) \oplus f_3(z_j)$.

We argue that with this strategy B's chance of forging one of the f_i's successfully is at least $\varepsilon/10q^2\log^2(q)$. Let Coll be the event that A finds a collision, and let Forge be the event that B successfully forges. We define 10 events $\mathsf{Ev}_1, \ldots, \mathsf{Ev}_{10}$ on A's query history such that $\mathsf{Coll} \implies \mathsf{Ev}_1 \vee \ldots \vee \mathsf{Ev}_{10}$. Then if Type_i denotes the event that B selects the Type i strategy, we show that $\Pr[\mathsf{Forge}|\mathsf{Ev}_i \wedge \mathsf{Type}_i] \geq 1/q^2\log^2(q)$ for $i = 1 \ldots 10$. This implies that $\Pr[\mathsf{Forge}] \geq \varepsilon/10q^2\log^2(q)$, since, by independence of the events

$\mathsf{Type}_1, \ldots, \mathsf{Type}_{10}$ from the events $\mathsf{Ev}_1, \ldots, \mathsf{Ev}_{10}$,

$$\begin{aligned}
\Pr[\mathsf{Forge}] &\geq \sum_{i=1}^{10} \Pr[\mathsf{Forge} \wedge \mathsf{Ev}_i \wedge \mathsf{Type}_i] \\
&= \sum_{i=1}^{10} \Pr[\mathsf{Ev}_i] \Pr[\mathsf{Type}_i] \Pr[\mathsf{Forge} | \mathsf{Ev}_i \wedge \mathsf{Type}_i] \\
&\geq \frac{1}{10q^2 \log^2(q)} \sum_{i=1}^{10} \Pr[\mathsf{Ev}_i] \\
&\geq \frac{\varepsilon}{10q^2 \log^2(q)}
\end{aligned}$$

The events $\mathsf{Ev}_1, \ldots, \mathsf{Ev}_{10}$ will be defined in terms of the following events $\mathsf{Win}_1, \ldots, \mathsf{Win}_{10}$:

- Win_1 is the event that A queries f_1 at distinct points x, x' such that $f_1(x) = f_1(x')$.
- Win_2 is the event that A queries f_2 at distinct points y, y' such that $f_2(y) = f_2(y')$.
- Win_3 is the event that A queries f_3 at distinct points z, z' such that $f_3(z) = f_3(z')$.
- Win_4 is the event that A queries f_3 at distinct points z, z' such that $z \oplus f_3(z) = z' \oplus f_3(z')$.
- Win_5 is the event that, when A is finished querying, there exists a $z \in \{0,1\}^n$ such that $|\{(x,y) : f_1(x) \oplus f_2(y)\}| > \log(q)$.
- Win_6 is the event that, when A is finished querying, there exists a $v \in \{0,1\}^n$ such that $|\{(x,z) : f_1(x) \oplus f_3(z)\}| > \log(q)$.
- Win_7 is the event that, when A is finished querying, there exists a $w \in \{0,1\}^n$ such that $|\{(y,z) : f_2(y) \oplus f_3(z)\}| > \log(q)$.
- Win_8 is the event that A obtains a collision such that the last query made by A to obtain the collision is a query to f_1.
- Win_9 is the event that A obtains a collision such that the last query made by A to obtain the collision is a query to f_2.
- Win_{10} is the event that A obtains a collision such that the last query made by A to obtain the collision is a query to f_3.

Now we define $\mathsf{Ev}_i = \mathsf{Win}_i \bigwedge \neg \left(\bigvee_{j<i} \mathsf{Win}_i \right)$ for $i = 1 \ldots 10$. Because $\mathsf{Coll} \implies \mathsf{Win}_8 \vee \mathsf{Win}_9 \vee \mathsf{Win}_{10}$ it is obvious that $\mathsf{Coll} \implies \mathsf{Ev}_1 \vee \cdots \vee \mathsf{Ev}_{10}$. We now show that $\Pr[\mathsf{Forge} | \mathsf{Ev}_i \wedge \mathsf{Type}_i] \geq 1/q^2 \log^2(q)$ for $i = 1 \ldots 10$.

Proposition 1 $\Pr[\mathsf{Forge} | \mathsf{Ev}_1 \wedge \mathsf{Type}_1] \geq 1/q^2$.

Proof. Say that B selects its Type 1 strategy and that A queries f_1 on two different points x, x' such that $f_1(x) = f_1(x')$. For its Type 1 strategy B selects $i < j$ uniformly in $\{1, \ldots, q\}$ and guesses that $f_1(x_i) = f_1(x_j)$ where x_i, x_j are the i-th and j-th query made by A to f_1. Then there is chance at least

$\binom{q}{2}^{-1} \geq 1/q^2$ that $\{x_i, x_j\} = \{x, x'\}$, in which case B's guess is correct, so $\Pr[\mathsf{Forge}|\mathsf{Ev}_1 \wedge \mathsf{Type}_1] \geq 1/q^2$. □

The following three propositions have similar proofs:

Proposition 2 $\Pr[\mathsf{Forge}|\mathsf{Ev}_2 \wedge \mathsf{Type}_2] \geq 1/q^2$.

Proposition 3 $\Pr[\mathsf{Forge}|\mathsf{Ev}_3 \wedge \mathsf{Type}_3] \geq 1/q^2$.

Proposition 4 $\Pr[\mathsf{Forge}|\mathsf{Ev}_4 \wedge \mathsf{Type}_4] \geq 1/q^2$.

Proposition 5 $\Pr[\mathsf{Forge}|\mathsf{Ev}_5 \wedge \mathsf{Type}_5] \geq 1/4q^2$.

Proof.

Let $S = (p_1, p_2, \ldots)$ be the sequence described in the Type 5 strategy, where each p_i is a pair (x, y) such that A has queried $f_1(x)$, $f_2(y)$. The value of a pair $p_i = (x, y)$ is $f_1(x) \oplus f_2(x)$, and B attempts to guess the value of the ℓ-th pair of the sequence at the moment when its value is about to be learned. For this B's strategy is to guess the answer uniformly among all numbers in $\{0,1\}^n$ that are already the values of at least k pairs in S, where k was selected randomly by B between 1 and $\log(q)$ at the start of the attack.

Each query by A adds a group of pairs to S. The pairs in a group either all have their x-coordinates or y-coordinates in common. When Ev_5 occurs f_1- and f_2-collisions do not occur (by definition of Ev_5, it cannot occur if Win_1 or Win_2) so the pairs in a group all have different values. We are now exactly in the case discussed under Case (2) in Section 4.2, where it is shown that B's chance of forgery is at least $1/4q^2$.

□

The following two propositions have the same proof as Propositions 5:

Proposition 6 $\Pr[\mathsf{Forge}|\mathsf{Ev}_6 \wedge \mathsf{Type}_6] \geq 1/4q^2$.

Proposition 7 $\Pr[\mathsf{Forge}|\mathsf{Ev}_7 \wedge \mathsf{Type}_7] \geq 1/4q^2$.

Proposition 8 $\Pr[\mathsf{Forge}|\mathsf{Ev}_8 \wedge \mathsf{Type}_8] \geq 1/q^2 \log^2(q)$.

Proof. Assume that the adversary obtains a collision where the last query needed to obtain the collision is a query x_0 made to f_1 and that $\neg\left(\bigvee_{i<7} \mathsf{Win}_i\right)$. Let (x_0, y_0), (x_1, y_1) be the two colliding inputs. If $x_0 = x_1$ then $f_1(x_0) = f_1(x_1)$ which implies $f_3(z_0) = f_3(z_1)$ where $z_0 = f_1(x_0) \oplus f_2(y_0)$ and $z_1 = f_1(x_1) \oplus f_3(z_1)$; then either $z_0 \neq z_1$ and Win_3, a contradiction, or $z_0 = z_1$ and $f_2(y_0) = f_2(y_1)$ but $y_0 \neq y_1$, so Win_2, another contradiction. Thus we can assume that $x_0 \neq x_1$. This implies the tuple (y_0, z_0, x_1, y_1) is in the set Q described in the Type 8 strategy. Note that $f_1(x_0) = f_2(y_0) \oplus z_0$.

With its Type 8 strategy B wins if it makes its guess on the query $f_1(x_0)$ and if it chooses a tuple $(y, z, x', y') \in Q$ such that $f_2(y) \oplus z = f_2(y_0) \oplus z_0$. Since B has chance $1/q$ of making its guess on the query $f_1(x_0)$ its chance of success is thus at least $1/q|Q'|$ where $Q' = \{(y, z) : (y, z, x', y') \in Q$ for some $x', y'\}$. Because $\neg\mathsf{Win}_5$ there are only at most $q\log(q)$ pairs (x', y') for which A has made the queries necessary to compute $F(x'\|y')$ (each such pair necessitates a query to f_3, and no query to f_3 validates more than $\log(q)$ pairs). Then because $\neg\mathsf{Win}_7$, the sum over these pairs (x', y') of the size of the set $\{(y, z) : (y, z, x', y') \in Q\}$ is at most $q\log^2(q)$, which implies $|Q'|$ is at most $q\log^2(q)$. Thus B has chance at least $1/q^2\log^2(q)$ of forging. □

Proposition 9 $\Pr[\mathsf{Forge}|\mathsf{Ev}_9 \wedge \mathsf{Type}_9] \geq 1/q^2 \log^2(q)$.

Proof. (Similar to Proposition 8.) Assume that the adversary obtains a collision where the last needed to obtain the collision is a query y_0 made to f_2 and that $\neg(\bigvee_{i<8} \mathsf{Win}_i)$. Let (x_0, y_0), (x_1, y_1) be the two colliding inputs. If $y_0 = y_1$ then $f_2(y_1) = f_2(y_1) = f_1(x_0) \oplus z_0 = f_1(x_1) \oplus z_1$ where $z_0 = f_1(x_0) \oplus f_2(y_0)$ and $z_1 = f_1(x_1) \oplus f_2(y_1)$. Adding the equation $f_1(x_0) \oplus f_3(z_0) = f_1(x_1) \oplus f_3(z_1)$ to $f_1(x_0) \oplus z_0 = f_1(x_1) \oplus z_1$ we obtain $z_0 \oplus f_3(z_0) = z_1 \oplus f_3(z_1)$. If $z_0 \neq z_1$ this is a contradiction to $\neg\mathsf{Win}_5$, otherwise $f_1(x_0) = f_1(x_1)$ and one can obtain a contradiction like in the proof of Proposition 8. Thus the tuple (x_0, z_0, x_1, y_1) is in the set R described in the Type 9 strategy.

With its Type 9 strategy B wins if it makes its guess on the query $f_2(y_0)$ and if it chooses a tuple $(x, z, x', y') \in R$ such that $f_1(x) \oplus z = f_1(x_0) \oplus z_0$. Since B has chance $1/q$ of making its guess on the query $f_2(y_0)$ its chance of success is thus at least $1/q|R'|$ where $R' = \{(x, z) : (x, z, x', y') \in R \text{ for some } x', y'\}$. Because $\neg\mathsf{Win}_5$ there are only at most $q \log(q)$ pairs (x', y') for which A has made the queries necessary to compute $F(x'\|y')$. Then because $\neg\mathsf{Win}_6$, the sum over these pairs (x', y') of the size of the set $\{(x, z) : (x, z, x', y') \in R\}$ is at most $q \log^2(q)$, which implies $|R'|$ is at most $q \log^2(q)$. Thus B has chance at least $1/q^2 \log^2(q)$ of forging. □

Proposition 10 $\Pr[\mathsf{Forge}|\mathsf{Ev}_{10} \wedge \mathsf{Type}_{10}] \geq 1/q^2 \log^2(q)$.

Proof. Assume that the adversary obtains a collision where the last query needed to obtain the collision is a query z_0 made to f_3 and that $\neg(\bigvee_{i<9} \mathsf{Win}_i)$. Let (x_0, y_0), (x_1, y_1) be the two colliding inputs, where $z_0 = f_1(x_0) \oplus f_2(y_0)$. As argued in Section 4.2, $z_1 := f_1(x_1) \oplus f_2(y_1)$ is distinct from z_0, because otherwise $f_1(x_0) = f_1(x_1)$ and consequently $f_2(y_0) = f_2(y_1)$, which would imply either Win_1 or Win_2, a contradiction. Thus B has chance $1/\binom{q}{2}$ that $z_i = z_0$ and $z_j = z_1$. To win B moreover needs to choose $(x_i, y_i) = (x_0, y_0)$, which B has chance $\geq 1/\log(q)$ since $\mathsf{Bin}(z_0) \leq \log(q)$ by $\neg\mathsf{Win}_5$, and likewise B has chance $\geq 1/\log(q)$ of choosing $(x_j, y_j) = (x_1, y_1)$. Thus B's chance of winning is at least $1/q^2 \log^2(q)$. □

C Security of SS as a MAC

In this section we investigate the MAC security of the Shrimpton-Stam (SS) compression function. We show that SS has MAC security εq^4 if the underlying block cipher has MAC security ε. Thus SS has *some* MAC security (in particular, cannot be trivially broken trivially like the compression function of CBC-MAC or enhanced CBC-MAC [8]), but not birthday security. Moreover, as we show with an artificially constructed MAC for which SS has security $\sim \varepsilon q^4$, the bound is essentially tight.

The results of this section underscore that the crucial ingredients of Theorem 2 are the MAC security of f_4 and the WCR security of SS, not the MAC security of SS itself. Our counterexample also shows a nice example of a compression function which is easier to forge than to find a collision for.

Theorem 5 *Let $f : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ and let $F = F[f_1, f_2, f_3]$ be the Shrimpton-Stam compression function. Then*

$$\mathbf{InSec}^{\mathrm{mac}}_F(t, q, 2qn) \leq 18q^4 \mathbf{InSec}^{\mathrm{mac}}_f(t + O(q^2 n), q, qn).$$

Proof. Let A be a MAC adversary for F achieving advantage ε running in time t and making q queries. We show there is a MAC adversary B for f running in time $t + O(q^2 n)$, making at most q queries and achieving advantage $\varepsilon/3q^4$.

A makes q queries $F(x_1, y_1)$, ..., $F(x_q, y_q)$ and then announces a forgery for $F(x_q + 1, y_q + 1)$. Let $z_i = f_1(x_i) + f_2(y_i)$ for $i <= q + 1$. As A makes queries, B computes the answers using its access to f_1, f_2, f_3, except for the moment when B decides to make its forgery. (We use the standard device that B uses its oracle f_k for one of the f_i's, choosing which one at random, while it samples two keys to

simulate the other two f_i's on its own; since only of the f_i's counts for valid forgeries, this introduces a factor $\frac{1}{3}$ into B's probability of success.)

B forges by using one of the following six types of strategies with equal probability:

Type 1: B anticipates that $F(x_q + 1, y_q + 1)$ will involve a fresh query to f_3, namely that z_{q+1} will be distinct from $z_1, \ldots, z_q$. Then B simply forgers f_3 on z_{q+1} by predicting the answer will be $w + f_1(x_{q+1})$. This strategy fails if z_{q+1} was already made as a query to f_3, succeeds otherwise. (So basically we can assume that z_{q+1} already appeared at some point as a query to f_3, or else B is successful.)

Type 2: B anticipates that $f_3(z_{q+1})$ was already made as a query to f_3 but that $f_1(x_{q+1})$ was not already made as a query to f_1, Then B forges by computing $f_2(y_{q+1})$ and guessing that $f_1(x_{q+1}) = f_2(x_{q+1}) + z_i$ for some randomly selected $i \leq q$. Then *if* $f_3(z_{q+1})$ was already made and $f_1(x_{q+1})$ hadn't already been made, B's chance of success is $1/q$ times A's chance of success (because there is $1/q$ chance of choosing the right z_i).

Type 3: symmetric to type 2, but anticipating that $f_2(y_{q+1})$ was not already made as a query instead of $f_1(x_{q+1})$.

For the remaining types, B anticipates that all three queries $f_1(x_{q+1})$, $f_2(y_{q+1})$ and $f_3(z_{q+1})$ will all have been made by the time A announces its forgery. Note that in this case, $z_{q+1} = f_1(x_i) + f_2(y_i)$ for some $i \leq q$, and either $x_i \neq x_{q+1}$ or $y_i \neq y_{q+1}$ (and possibly both). To cover the cases, B play three different types:

Type 4: B anticipates that $x_i = x_{q+1}$ and $y_i \neq y_{q+1}$. Then since the query $f_2(y_{q+1})$ was already made there is some $j \leq q$, $j \neq i$, such that $y_j = y_{q+1}$. So $f_2(y_i) = z_i + f_1(x_i) = z_{q+1} + f_1(x_{q+1}) = f_2(y_{q+1}) = f_2(y_j)$. Therefore there is a collision in f_2, and B does its forgery by guessing a collision for f_2 (with probability $1/q^2$ of success if a collision does occur).

Type 5: B anticipates that $y_i = y_{q+1}$ and $x_i \neq x_{q+1}$. Symmetric to type 4, B guesses a collision for f_1.

Type 6: B anticipates that $x_i \neq x_{q+1}$, $y_i \neq y_{q+1}$. Let j be the smallest index such that $x_j = x_{q+1}$, let h be the smallest index such that $y_h = y_{q+1}$. Also let m be the smallest index such that $x_m = x_i$ and b be the smallest index such that $y_b = y_i$. Then $f_1(x_m) + f_2(y_b) = f_1(x_j) + f_2(y_h)$, so any one of the four values $f_1(x_m)$, $f_2(y_b)$, $f_1(x_j)$, $f_2(y_h)$ can be inferred from the three others. Also note that $m \neq j$ and $b \neq h$, since we are assuming $x_i \neq x_{q+1}$ and $y_i \neq y_{q+1}$. Then B guesses the sets $\{m, j\}$ and $\{h, b\}$ (two distinct integers each) and does the obvious forgery on the query corresponding to the highest of the four integers $\{m, j, h, b\}$. For example, if b is the highest, B guesses that $f_2(y_b) = f_1(x_m) + f_2(y_h) + f_1(x_j)$. It is possible, say, that two of the integers $\{m, j, h, b\}$ are both highest (say $j = b$), in which case B has the choice of which of those two queries to make first and which to forge (if $j = b$, say, B could compute $f_1(x_j)$ and forge $f_2(y_b)$ or compute $f_2(y_b)$ and forge $f_1(x_j)$). The chance of guessing the four indices correctly (conditioned on the assumptions of this strategy) is $1/q^4$.

B's overall chance of success of $p/18q^4$ follows from the fact that one of the six scenarios anticipated by B in its strategies must occur, and the worst chance it has of winning for any scenario is $1/q^4$, if that scenario occurs and it is playing the appropriate game, coupled with the fact that a successful forgery is only a forgery for the real f_k with probability $\frac{1}{3}$. □

We now show that Theorem 5 is essentially tight by showing there exists an (artificial) function family f for which $\mathbf{InSec}^{\mathrm{mac}}_F(t, q, 2qn) \geq 4q^4\mathbf{InSec}^{\mathrm{mac}}_f(t + O(q^2n), q, qn)/27$. We assume that f is an n-bit to n-bit block cipher where $n = 3k$ for some k.

We individually design the functions f_1, f_2, f_3, with f_1 and f_2 sharing the same design and f_3 having a separate design. Since these functions are in fact part of a common block cipher, we devote 2/3 of the keys for functions with the behavior of f_1, f_2 and 1/3 of the keys for functions with the behavior of f_3. An instantiation of F uses random keys, so there is chance 4/27 that the key for each f_i comes for the "correct" group, which affects the probability of finding a collision by that much (namely, we think of the MAC-forging adversary as abandoning if f_1, f_2, f_3 do not each have keys coming from the correct group).

The functions f_1 and f_2 fill the first k bits of the n-bit output with a random value and the bottom $2k$ bits with 0's. The functions f_3 fills the first k bits with 0's, the middle k bits with its first k bits of its input, and the last k bits with a perfect random value.

Note that if $k = n/3$ is sufficiently large, each f_i is a good MAC, with constant security $\mathbf{InSec}_f^{\mathrm{mac}}(T, q, qn) = 1/2^k$ against chosen message attack (independently of T and of q).

With this definition of the f_i's the output $F(x, y)$ of SS is the first k bits for $f_1(x)$ followed by the first k bits of $f_1(x) + f_2(y)$ followed by the last k bits of $f_3(f_1(x) + f_2(y))$. Note that by seeing this output one can reconstruct the values $f_1(x)$, $f_2(y)$ and $f_3(f_1(x) + f_2(y))$. Namely if $F(x, y) = w = t|u|v$ where t, u, v are k-bit values, one has

$$\begin{aligned} f_1(x) &= t|0|0 && \text{(where each 0 is } k \text{ bits)} \\ f_2(y) &= t + u|0|0 && \text{(because } u \text{ is the first } k \text{ bits of } f_1(x) + f_2(y)) \\ f_3(f_1(x) + f_2(y)) &= 0|u|v \end{aligned}$$

The adversary A makes random queries $F(x_1, y_1)$, $F(x_2, y_2)$, ... to its oracle until it finds x_i, x_j, y_m, y_h such that $f_1(x_i) + f_2(y_m) = f_1(x_j) + f_2(y_h)$ and such that j is not equal to h (recall that the adversary knows $f_1(x_i)$ and $f_2(y_i)$ from $F(x_i, y_i)$). On average, because f_1 and f_2 only have k bits of active output, the adversary's chance of finding such a quadruple is $q^4/2^k$. Once the adversary has such a quadruple it makes the query $F(x_i, y_m)$ if the query hasn't already been made (namely, if i is not equal to m). Say $F(x_i, y_m) = t_1|u_1|v_1$. Then the adversary forges by predicting that $F(x_j, y_h) = t_2|u_1|v_1$ where t_2 is the first k bits of output of $f_1(x_j)$ (which are known to the adversary). Obviously this forgery succeeds with probability 1, and the overall chance of success is $q^4/2^{n/3} = q^4\mathbf{InSec}_f^{\mathrm{mac}}(t + O(q^2 n), q, qn)$. We then obtain the result by taking into account the factor 4/27 mentioned above.