Variable Tag Length Message Authentication Code Schemes

Sebati Ghosh and Palash Sarkar
Indian Statistical Institute
203, B.T.Road, Kolkata, India - 700108.
{sebati_r, palash}@isical.ac.in

November 22, 2019

Abstract

This work studies message authentication code (MAC) schemes supporting variable tag lengths. We provide a formalisation of such a scheme. Several variants of the classical Wegman-Carter MAC scheme are considered. Most of these are shown to be insecure by pointing out detailed attacks. One of these schemes is highlighted and proved to be secure. We further build on this scheme to obtain single-key nonce-based variable tag length MAC schemes utilising either a stream cipher or a short-output pseudo-random function. These schemes can be efficiently instantiated using practical well known primitives. We further consider the problem of building variable tag length MAC schemes without nonces. Again, efficient constructions of such schemes are described along with their proofs of security.
Keywords: MAC, variable tag length, Wegman-Carter, security bound.

1 Introduction

Message authentication code (MAC) is the cryptographic mechanism to ensure the authenticity of messages transmitted across a public channel. A MAC scheme typically appends a short length tag to the message which is then transmitted. At the receiving end, a verification algorithm is run on the message-tag pair to confirm the authenticity. In such a set-up, the sender and the receiver share a previously agreed upon secret key.

Most MAC schemes specify a single value for the tag length. The question that we address in this work is the following. Is it possible to have MAC schemes where the tag length can vary? While the question seems to be a natural one, there does not appear to have been much discussion about this issue in the literature. The only material we could locate is an almost 15-year old CFRG [23] discussion pertaining to different tag lengths suggested for the MAC scheme UMAC [14]. This scheme had the possibility of using 32-bit, 64-bit, 96-bit and 128-bit tags. Finney [11], crediting "Dan Bernstein's poly1305-aes mailing list", had pointed out that this feature would allow forging a 64-bit tag using about 2^{33} queries. A later post [12] explains the issue further and suggests how a valid 128-bit tag can be obtained with only about 2^{34} queries. Wagner [24] supporting the issue raised by Finney, had mentioned that to fix the problem "it suffices to ensure that the tag length is a parameter that is immutably bound to the key and never changed. In other words, never use the same key with different parameter sizes." Following this suggestion, Section 6.5 of the UMAC specification [14] states that a "UMAC key (or session) must have an associated and immutable tag length". Another suggestion put forward by Finney [12] to handle the issue requires "stealing two bits of input into the block cipher from the nonce and using them to encode tag size". Apart

from the interesting discussion on variable tag lengths for the UMAC scheme, we know of no other place where the issue of variable tag length MAC schemes has been considered.

The question of variable tag length received some attention in the past few years in the context of authenticated encryption (AE) schemes and the CAESAR [9] competition. Manger [16] pointed out that for the AE scheme OCB, 64-bit, 96-bit and 128-bit tags are defined where the "64-bit and 96-bit tags are simply truncated 128-bit tags". This leads to simple truncation attacks on the scheme. An earlier paper by Rogaway and Wagner [21] had also discussed the problem of variable tag lengths in the context of the AE scheme CCM. A formal treatment of variable tag length AE schemes has been given by Reyhanitabar, Vaudenay and Vižar [20].

Two concrete motivations are provided in [20] as to why a variable tag length AE scheme may indeed be desirable in practice. The first mentions that variable tag lengths may be used with the same key due to "misuse and poorly engineered security systems". The second reason is that for resource constrained devices, variable tag lengths may be desirable though changing the key for every tag length may be infeasible due to limited bandwidth and low power.

While the above two reasons have been put forward in the context of AE schemes, they are equally valid for MAC schemes. More generally, the issue of "mis-implementation" (also called "footguns") [19] of cryptographic primitives has been extensively discussed as part of the discussion forum on post-quantum cryptography.

More concretely, Auth256 [7] is a Wegman-Carter type construction targeted at the 256-bit security level. Similarly, a 256-bit secure universal hash function has been proposed in [10], which can be mated to a 256-bit secure PRF using the Wegman-Carter template to obtain a 256-bit secure MAC. Such MAC schemes would be appropriate for high-security applications, or, for a post-quantum world. On the other hand, bandwidth limited applications would require shorter tags. Also, the possibility of mis-implementation using tag truncation remains. So, the question of designing a MAC scheme which can support various tag lengths up to 256 bits is of practical interest.

To summarise, the problem of variable tag length MAC schemes has been briefly mentioned about 15 years ago. Since then, there has neither been any formal treatment of the topic and nor has there been any variable tag length MAC scheme which is accompanied by a proof of security. The problem of constructing such MAC schemes, though, is of contemporary and future practical interest.

Our Contributions

We provide a formalisation of the notion of security for a variable tag length MAC scheme. For the same key, the desired tag length is to be provided as part of the input to the tag generation algorithm. Consequently, in the security model, we allow the adversary to control the tag length as well as the message. This is an extension of the usual security model for MAC schemes. We provide a formalisation of the new model both for the case of MAC schemes with and without nonces.

We consider the problem of obtaining secure variable tag length MAC schemes. The Wegman-Carter [26] scheme is the classical nonce-based MAC scheme. A naive approach to obtain a variable tag length MAC scheme is to truncate tags produced by the Wegman-Carter scheme. We show an easy attack on such a truncation scheme. Next, we consider eight possible "natural" variants that arise from the Wegman-Carter MAC scheme. We show attacks on six of these schemes. Among the attacked schemes is the scheme obtained by nonce stealing following the suggestion of Finney [11] as mentioned above. One of the eight schemes is generically secure since it uses independent keys for different tag lengths. The last of the eight schemes is proved to be secure. This scheme uses

nonce stealing *but*, for different tag lengths, it uses independent keys for the universal hash function component of the Wegman-Carter scheme.

From a practical point of view, it is desirable to have a scheme which uses a single key. The key for the hash function is then derived from the key of the scheme and the tag length. The manner in which such derivation is made depends upon the primitive used to derive the hash key. We show two methods of deriving the hash key. The first method uses a stream cipher while the second method uses a short output length pseudo-random function (PRF). So, in effect, we obtain two constructions of single key variable tag length MAC scheme with nonces.

Constructions of variable tag length MAC schemes which do not use nonces have also been provided. We show that the availability of a PRF scheme which supports variable input lengths directly leads to a variable tag length MAC scheme. Three constructions of variable input length PRF schemes are presented and analysed.

All the schemes that we describe can be instantiated by readily available concrete cryptographic primitives. For example, either of the 256-bit secure universal hash functions in [7, 10] can be combined with Salsa20 [3] to obtain nonce-based MAC schemes supporting variable tag lengths up to 256 bits. So, our work provides templates for designing efficient and practical MAC schemes which support variable tag lengths.

AE with variable tag length versus MAC with variable tag length: An AE scheme supporting variable tag length has been proposed in [20]. Given a message M, suppose that the ciphertext is (C, tag). One may wonder whether we can construct a MAC scheme by throwing away C and keeping tag. A MAC scheme obtained from the AE scheme in [20] in this manner is not secure. A simple reshuffling of the message blocks will give rise to the same tag. This, of course, has no implication to the security of the construction in [20] as an AE scheme. More generally, the above kind of simple strategy will fail to produce a secure MAC scheme from a secure AE scheme.

Previous and Related Works

The notion of MAC is several decades old. So, there is an extensive literature on this topic. Here we mention the papers which are directly related to our work.

The Wegman-Carter [26] scheme is four decades old. Several important and practical MAC schemes, such as UMAC [8] and Poly1305 [4] are based on the Wegman-Carter scheme. From a theoretical point of view, the security of the Wegman-Carter scheme was later analysed by Shoup [22] and Bernstein [5]. Recently, the optimality of Bernstein's bound was established in [15, 18].

2 Definitions

Let x be a binary string: $\mathsf{len}(x)$ denotes the length of x; for a non-negative integer λ, $\mathsf{msb}_\lambda(x)$ denotes the λ most significant bits of x. Given an integer i in the range $0 \leq i < 2^k - 1$, $\mathsf{bin}_k(i)$ denotes the k-bit binary representation of i.

Throughout this paper, n is a fixed positive integer.

2.1 Hash Function

Let $\mathcal{M}$ and Θ be finite non-empty sets. Let $\{H_\tau\}_{\tau \in \Theta}$ be an indexed family of functions such that for each $\tau \in \Theta$, $H_\tau : \mathcal{M} \to \{0,1\}^n$. The sets $\mathcal{M}$ and Θ are respectively the message and the key

spaces. Typically, a message is a binary string of some maximum length.

For distinct $x, x' \in \mathcal{M}$ and any n-bit string y, the differential probability of H_τ for the triplet (x, x', y) is defined to be $\Pr_\tau[H_\tau(x) \oplus H_\tau(x') = y]$, where the probability is taken over the uniform random choice of τ from Θ. The differential probability may depend on the lengths of x and x'. Suppose L is the maximum of the lengths of the binary strings in $\mathcal{M}$. Let $\varepsilon : \{0, \ldots, L\}^2 \to [0, 1]$ be a function such that the differential probability for any (x, x', y) is at most $\varepsilon(\mathsf{len}(x), \mathsf{len}(x'))$. Then the family $\{H_\tau\}_{\tau \in \Theta}$ is said to be ε-AXU.

2.2 Pseudo-Random Function

Let $\mathcal{D}$ and $\mathcal{R}$ be finite non-empty sets of binary strings and $\mathcal{K}$ be a finite non-empty set. Let $\{F_K\}_{K \in \mathcal{K}}$ be a keyed family of functions with $F_K : \mathcal{D} \to \mathcal{R}$. Informally speaking, the function family $\{F_K\}_{K \in \mathcal{K}}$ is considered to be pseudo-random if a resource limited adversary is unable to distinguish it from a uniform random function from $\mathcal{D}$ to $\mathcal{R}$. This is formalised in the following manner.

We consider an adversary $\mathcal{A}$ which has access to an oracle $\mathcal{O}$, which is written as $\mathcal{A}^{\mathcal{O}}$. $\mathcal{A}$ adaptively sends queries to $\mathcal{O}$ and receives appropriate responses. At the end of the interaction, $\mathcal{A}$ outputs a bit. The adversary is allowed to perform computations and also has access to private random bits.

Let $(K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{F_K(\cdot)} \Rightarrow 1)$ denote the event that K is chosen uniformly at random from $\mathcal{K}$ and the adversary produces 1 after interacting with the oracle $F_K(\cdot)$. Let $\$(\cdot)$ be a function chosen uniformly at random from the set of all functions from $\mathcal{D}$ to $\mathcal{R}$. Let $(\mathcal{A}^{\$(\cdot)} \Rightarrow 1)$ denote the event that the adversary produces 1 after interacting with the oracle $\$(\cdot)$.

The advantage of $\mathcal{A}$ in breaking the pseudo-randomness of $\{F_K\}_{K \in \mathcal{K}}$ is defined as follows.

$$\mathsf{Adv}_F^{\mathrm{prf}}(\mathcal{A}) = \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{F_K(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot)} \Rightarrow 1\right]. \tag{1}$$

The probabilities are over the randomness of $\mathcal{A}$, the choice of K and the randomness of $\$(\cdot)$.

Suppose that $\mathcal{A}$ makes a total of q queries sending a total of σ bits in all the queries. By $\mathsf{Adv}_F^{\mathrm{prf}}(t, q, \sigma)$ we will denote the maximum advantage of any adversary taking time at most t, making at most q queries and sending at most σ bits in all its queries.

2.3 Variable Tag Length Message Authentication Code

A MAC scheme has two algorithms, namely, the tag generation algorithm and the verification algorithm. Typically, in a MAC scheme, tags are binary strings of some fixed length. The definition of MAC schemes, however, does not require tags to have the same length. So, it is possible to consider variable length tags within the ambit of the currently used definition of MAC schemes.

Our goal, on the other hand, is different. We would like the tag length to be provided as part of the input to the tag generation and verification algorithms. So, for the same message, by providing different values of the tag length, it is possible to generate tags of different lengths. This feature is not covered by the presently used definition of MAC schemes. We extend the syntax of MAC schemes and the definition of security to incorporate this feature.

A MAC scheme is given by the message space $\mathcal{M}$, the key space $\mathcal{K}$, the allowed set $\mathcal{L}$ of tag lengths, the tag space $\mathcal{T}$; and the two algorithms $\mathsf{MAC.Gen}(K, x, \lambda)$ and $\mathsf{MAC.Verify}(K, x, \mathsf{tag}, \lambda)$, where $K \in \mathcal{K}$, $x \in \mathcal{M}$, $\lambda \in \mathcal{L}$ and $\mathsf{tag} \in \mathcal{T}$. We consider $\mathcal{M}$, $\mathcal{K}$ and $\mathcal{L}$ to be finite non-empty sets and $\mathcal{T}$ to be equal to $\cup_{i \in \mathcal{L}}\{0,1\}^i$. We write $\mathsf{MAC.Gen}_K(x, \lambda)$ to denote $\mathsf{MAC.Gen}(K, x, \lambda)$, and $\mathsf{MAC.Verify}_K(x, \mathsf{tag}, \lambda)$ to denote $\mathsf{MAC.Verify}(K, x, \mathsf{tag}, \lambda)$.

The inputs and outputs of $\mathsf{MAC.Gen}_K(x,\lambda)$ and $\mathsf{MAC.Verify}_K(x,\mathsf{tag},\lambda)$ are as follows.

- $\mathsf{MAC.Gen}_K(x,\lambda)$:
  input: $K \in \mathcal{K}$; $x \in \mathcal{M}$; and $\lambda \in \mathcal{L}$.
  output: $\mathsf{tag} \in \mathcal{T}$ is a binary string of length λ.

- $\mathsf{MAC.Verify}_K(x,\mathsf{tag},\lambda)$:
  input: $K \in \mathcal{K}$; $x \in \mathcal{M}$; $\mathsf{tag} \in \mathcal{T}$; and $\lambda \in \mathcal{L}$ such that tag is of length λ.
  output: an element from the set $\{\mathsf{true},\mathsf{false}\}$. The value true indicates that the input is accepted while the value false indicates that the input is rejected.

The following correctness condition must hold.

$$\mathsf{MAC.Verify}_K(x,\mathsf{MAC.Gen}_K(x,\lambda),\lambda) \quad = \quad \mathsf{true}.$$

Security: The security for a MAC scheme against an adversary $\mathcal{A}$ is modelled as follows. Suppose K is chosen uniformly at random from $\mathcal{K}$ and the tag generation and verification algorithms are instantiated with K. $\mathcal{A}$ is given oracle access to the tag generation and the verification algorithms. This means that $\mathcal{A}$ can adaptively query the tag generation algorithm with inputs of the type (x,λ) and gets back in response the output of $\mathsf{MAC.Gen}_K(x,\lambda)$. Also, $\mathcal{A}$ adaptively queries the verification algorithm with inputs of the type (x,tag,λ) and gets back in response either true or false. Further, $\mathcal{A}$ can interleave queries to the tag generation and the verification oracles.

The adversary makes a total of q_g queries to the tag generation algorithm and a total of q_v queries to the verification algorithm. Let the queries to the tag generation algorithm be

$$\left(x_g^{(1)},\lambda_g^{(1)}\right),\ldots,\left(x_g^{(q_g)},\lambda_g^{(q_g)}\right)$$

and the corresponding responses be $\mathsf{tag}_g^{(1)},\ldots,\mathsf{tag}_g^{(q_g)}$ respectively. Similarly, let the queries to the verification algorithm be

$$\left(x_v^{(1)},\mathsf{tag}_v^{(1)},\lambda_v^{(1)}\right),\ldots,\left(x_v^{(q_v)},\mathsf{tag}_v^{(q_v)},\lambda_v^{(q_v)}\right)$$

and the corresponding responses be $\mathsf{xxx}_v^{(1)},\ldots,\mathsf{xxx}_v^{(q_v)}$ respectively, where for $1 \le j \le q_v$, $\mathsf{xxx}_v^{(j)}$ is either true or false. The query profile of $\mathcal{A}$ is the list

$$\mathfrak{C} \quad = \quad \left(q_g,q_v,(\mathfrak{m}_g^{(1)},\lambda_g^{(1)}),\ldots,(\mathfrak{m}_g^{(q_g)},\lambda_g^{(q_g)}),(\mathfrak{m}_v^{(1)},\lambda_v^{(1)}),\ldots,(\mathfrak{m}_v^{(q_v)},\lambda_v^{(q_v)})\right) \tag{2}$$

where for $1 \le s \le q_g$, $\mathfrak{m}_g^{(s)} = \mathsf{len}(x_g^{(s)})$ and for $1 \le s \le q_v$, $\mathfrak{m}_v^{(s)} = \mathsf{len}(x_v^{(s)})$.

The restriction on the adversary is that it should not make any useless query. A query is useless if its response can be computed by the adversary. This means that the adversary should not repeat a query to the tag generation oracle or the verification oracle; and it should not query the verification oracle with $\left(x_g^{(i)},\mathsf{tag}_g^{(i)},\lambda_g^{(i)}\right)$ for any i in $\{1,\ldots,q_g\}$.

For $\lambda \in \mathcal{L}$, let $\mathsf{succ}_{\mathcal{A}}(\lambda)$ be the event that there is a $j \in \{1,\ldots,q_v\}$ such that $\lambda_v^{(j)} = \lambda$ and $\mathsf{MAC.Verify}_K\left(x_v^{(j)},\mathsf{tag}_v^{(j)},\lambda_v^{(j)}\right)$ returns true. For each $\lambda \in \mathcal{L}$, the adversary's advantage in breaking the authenticity of MAC is defined to be $\Pr[\mathsf{succ}_{\mathcal{A}}(\lambda)]$. This is written as follows.

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{MAC}}[\lambda](\mathcal{A}) \quad = \quad \Pr\left[\mathsf{succ}_{\mathcal{A}}(\lambda)\right]. \tag{3}$$

The above probability is taken over the uniform random choice of K from $\mathcal{K}$ and over the possible internal randomness of the adversary $\mathcal{A}$.

The query complexity is the total number of bits sent by the adversary in all its queries. For tag generation queries, this consists of both the number of bits sent as part of the messages and the number of bits sent as part of λ_g's. For verification queries, this consists of the number of bits sent as part of the messages, the number of bits sent as part of the tags and the number of bits sent as part of λ_v's. Let the q_g tag generation queries require a total of σ_g bits and the q_v verification queries require a total of σ_v bits. So, $\sigma_g = \sum_{1 \leq i \leq q_g} (\mathsf{len}(x_g^{(i)}) + \mathsf{len}(\lambda_g^{(i)})) = \sum_{1 \leq i \leq q_g} (\mathfrak{m}_g^{(i)} + \mathsf{len}(\lambda_g^{(i)}))$ and $\sigma_v = \sum_{1 \leq i \leq q_v} (\mathsf{len}(x_v^{(i)}) + \mathsf{len}(\mathsf{tag}_v^{(i)}) + \mathsf{len}(\lambda_v^{(i)})) = \sum_{1 \leq i \leq q_v} (\mathfrak{m}_v^{(i)} + \lambda_v^{(i)} + \mathsf{len}(\lambda_v^{(i)}))$, as $\mathsf{len}(\mathsf{tag}_v^{(i)}) = \lambda_v^{(i)}$. If the elements of $\mathcal{L}$ are expressed as $\mathfrak{t}$-bit binary strings, then $\sigma_g = \sum_{1 \leq i \leq q_g} \mathfrak{m}_g^{(i)} + q_g \mathfrak{t}$ and $\sigma_v = \sum_{1 \leq i \leq q_v} (\mathfrak{m}_v^{(i)} + \lambda_v^{(i)}) + q_v \mathfrak{t}$.

Given a query profile $\mathfrak{C}$, $\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{MAC}}[\lambda](t, \mathfrak{C})$ is the maximum of $\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{MAC}}[\lambda](\mathcal{A})$ taken over all adversaries $\mathcal{A}$ running in time t and having query profile $\mathfrak{C}$.

Information theoretic security: This consists of analysing the security of a MAC scheme against a computationally unbounded adversary. In other words, the probability in (3) is considered for an adversary $\mathcal{A}$ without any reference to the run time of $\mathcal{A}$. For such a computationally unbounded adversary $\mathcal{A}$, without loss of generality, we may assume $\mathcal{A}$ to be deterministic. In the context of information theoretic security, given a query profile $\mathfrak{C}$, $\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{MAC}}[\lambda](\mathfrak{C})$ is the maximum of $\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{MAC}}[\lambda](\mathcal{A})$ taken over all adversaries $\mathcal{A}$ having query profile $\mathfrak{C}$.

2.4 Variable Tag Length Nonce-Based Message Authentication Code

In this section, we extend the notion of a MAC scheme described in the previous section to that of a MAC scheme supporting a nonce.

A nonce-based MAC scheme is given by the message space $\mathcal{M}$, the nonce space $\mathcal{N}$, the key space $\mathcal{K}$, the allowed set $\mathcal{L}$ of tag lengths, the tag space $\mathcal{T}$; and two algorithms $\mathsf{NMAC.Gen}(K, N, x, \lambda)$ and $\mathsf{NMAC.Verify}(K, N, x, \mathsf{tag}, \lambda)$, where $K \in \mathcal{K}$, $N \in \mathcal{N}$, $x \in \mathcal{M}$, $\lambda \in \mathcal{L}$ and $\mathsf{tag} \in \mathcal{T}$. We consider $\mathcal{M}$, $\mathcal{N}$, $\mathcal{K}$ and $\mathcal{L}$ to be finite non-empty sets and $\mathcal{T}$ to be equal to $\cup_{i \in \mathcal{L}} \{0,1\}^i$. We write $\mathsf{NMAC.Gen}_K(N, x, \lambda)$ to denote $\mathsf{NMAC.Gen}(K, N, x, \lambda)$, and $\mathsf{NMAC.Verify}_K(N, x, \mathsf{tag}, \lambda)$ to denote $\mathsf{NMAC.Verify}(K, N, x, \mathsf{tag}, \lambda)$.

The inputs and outputs of $\mathsf{NMAC.Gen}_K(N, x, \lambda)$ and $\mathsf{NMAC.Verify}_K(N, x, \mathsf{tag}, \lambda)$ are as follows.

- $\mathsf{NMAC.Gen}_K(N, x, \lambda)$:
  input: $K \in \mathcal{K}$; $N \in \mathcal{N}$; $x \in \mathcal{M}$; and $\lambda \in \mathcal{L}$.
  output: $\mathsf{tag} \in \mathcal{T}$ is a binary string of length λ.

- $\mathsf{NMAC.Verify}_K(N, x, \mathsf{tag}, \lambda)$:
  input: $K \in \mathcal{K}$; $N \in \mathcal{N}$; $x \in \mathcal{M}$; $\mathsf{tag} \in \mathcal{T}$; and $\lambda \in \mathcal{L}$ such that tag is of length λ.
  output: an element from the set $\{\mathsf{true}, \mathsf{false}\}$. The value true indicates that the input is accepted while the value false indicates that the input is rejected.

The following correctness condition must hold.

$$\mathsf{NMAC.Verify}_K(N, x, \mathsf{NMAC.Gen}_K(N, x, \lambda), \lambda) \quad = \quad \mathsf{true}.$$

Security: The security for a nonce-based MAC scheme against an adversary $\mathcal{A}$ is modelled in a manner similar to that of a MAC scheme. As before, suppose K is chosen uniformly at random from $\mathcal{K}$ and the tag generation and verification algorithms are instantiated with K. $\mathcal{A}$ is given oracle access to the tag generation and the verification algorithms. $\mathcal{A}$ makes a total of q_g queries to the tag generation oracle and a total of q_v queries to the verification oracle. The queries are made adaptively and queries to the tag generation oracle can be interleaved with those to the verification oracle.

Let the queries to the tag generation oracle be

$$\left(N_g^{(1)}, x_g^{(1)}, \lambda_g^{(1)}\right), \ldots, \left(N_g^{(q_g)}, x_g^{(q_g)}, \lambda_g^{(q_g)}\right)$$

and the corresponding responses be $\mathsf{tag}_g^{(1)}, \ldots, \mathsf{tag}_g^{(q_g)}$ respectively. Similarly, let the queries to the verification oracle be

$$\left(N_v^{(1)}, x_v^{(1)}, \mathsf{tag}_v^{(1)}, \lambda_v^{(1)}\right), \ldots, \left(N_v^{(q_v)}, x_v^{(q_v)}, \mathsf{tag}_v^{(q_v)}, \lambda_v^{(q_v)}\right)$$

and the corresponding responses be $\mathsf{xxx}_v^{(1)}, \ldots, \mathsf{xxx}_v^{(q_v)}$ respectively, where for $1 \leq j \leq q_v$, $\mathsf{xxx}_v^{(j)}$ is either true or false. The query profile of $\mathcal{A}$ is the list

$$\begin{aligned} \mathfrak{C} \;=\; & (q_g, q_v, (\mathfrak{n}_g^{(1)}, \mathfrak{m}_g^{(1)}, \lambda_g^{(1)}), \ldots, (\mathfrak{n}_g^{(q_g)}, \mathfrak{m}_g^{(q_g)}, \lambda_g^{(q_g)}), (\mathfrak{n}_v^{(1)}, \mathfrak{m}_v^{(1)}, \lambda_v^{(1)}), \\ & \qquad \ldots, (\mathfrak{n}_v^{(q_v)}, \mathfrak{m}_v^{(q_v)}, \lambda_v^{(q_v)})) \end{aligned} \tag{4}$$

where for $1 \leq s \leq q_g$, $\mathfrak{n}_g^{(s)} = \mathsf{len}(N_g^{(s)}), \mathfrak{m}_g^{(s)} = \mathsf{len}(x_g^{(s)})$ and for $1 \leq s \leq q_v$, $\mathfrak{n}_v^{(s)} = \mathsf{len}(N_v^{(s)}), \mathfrak{m}_v^{(s)} = \mathsf{len}(x_v^{(s)})$.

There are two restrictions on the adversary. The first is a weaker form of nonce-respecting behaviour, namely, $\left(N_g^{(i)}, \lambda_g^{(i)}\right) \neq \left(N_g^{(j)}, \lambda_g^{(j)}\right)$ for $1 \leq i < j \leq q_g$. Note that the adversary is allowed to repeat (nonce, tag-length) pair for verification queries and it is also allowed to re-use a (nonce, tag-length) pair used in a tag generation query in one or more verification queries. The second restriction on the adversary is that it should not make any useless query. As before, a query is useless if its response can be computed by the adversary. This means that the adversary should not repeat a query to the tag generation oracle or the verification oracle; and it should not query the verification oracle with $\left(N_g^{(i)}, x_g^{(i)}, \mathsf{tag}_g^{(i)}, \lambda_g^{(i)}\right)$ for any i in $\{1, \ldots, q_g\}$.

For $\lambda \in \mathcal{L}$, let $\mathsf{succ}_{\mathcal{A}}(\lambda)$ be the event that there is some $j \in \{1, \ldots, q_v\}$ such that $\lambda_v^{(j)} = \lambda$ and $\mathsf{NMAC.Verify}_K\left(N_v^{(j)}, x_v^{(j)}, \mathsf{tag}_v^{(j)}, \lambda_v^{(j)}\right)$ returns true. For each $\lambda \in \mathcal{L}$, the adversary's advantage in breaking the authenticity of NMAC is defined to be $\Pr[\mathsf{succ}_{\mathcal{A}}(\lambda)]$. This is written as follows.

$$\mathsf{Adv}_{\mathsf{NMAC}}^{\mathrm{auth}}[\lambda](\mathcal{A}) \;=\; \Pr\left[\mathsf{succ}_{\mathcal{A}}(\lambda)\right]. \tag{5}$$

The above probability is taken over the uniform random choice of K from $\mathcal{K}$ and over the possible internal randomness of the adversary $\mathcal{A}$.

The query complexity is the total number of bits sent by the adversary in all its queries. For tag generation queries, this consists of the number of bits sent as part of the nonces, the messages and the λ_g's. For verification queries, this consists of the number of bits sent as part of the nonces, the messages, the tags and the λ_v's. Let the q_g tag generation queries require a total of σ_g bits and the q_v verification queries require a total of σ_v bits. So, $\sigma_g = \sum_{1 \leq i \leq q_g} (\mathsf{len}(N_g^{(i)}) + \mathsf{len}(x_g^{(i)}) + \mathsf{len}(\lambda_g^{(i)})) =$

$\sum_{1\le i\le q_g}(\mathfrak{n}_g^{(i)} + \mathfrak{m}_g^{(i)} + \mathsf{len}(\lambda_g^{(i)}))$ and $\sigma_v = \sum_{1\le i\le q_v}(\mathsf{len}(N_v^{(i)}) + \mathsf{len}(x_v^{(i)}) + \mathsf{len}(\mathsf{tag}_v^{(i)}) + \mathsf{len}(\lambda_v^{(i)})) = \sum_{1\le i\le q_v}(\mathfrak{n}_v^{(i)} + \mathfrak{m}_v^{(i)} + \lambda_v^{(i)} + \mathsf{len}(\lambda_v^{(i)}))$, as $\mathsf{len}(\mathsf{tag}_v^{(i)}) = \lambda_v^{(i)}$. If the elements of $\mathcal{L}$ are expressed as $\mathfrak{t}$-bit binary strings, then $\sigma_g = \sum_{1\le i\le q_g}(\mathfrak{n}_g^{(i)} + \mathfrak{m}_g^{(i)}) + q_g\mathfrak{t}$ and $\sigma_v = \sum_{1\le i\le q_v}(\mathfrak{n}_v^{(i)} + \mathfrak{m}_v^{(i)} + \lambda_v^{(i)}) + q_v\mathfrak{t}$.

Given a query profile $\mathfrak{C}$, $\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{NMAC}}[\lambda](t, \mathfrak{C})$ is the maximum of $\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{NMAC}}[\lambda](\mathcal{A})$ taken over all adversaries running in time t and having query profile $\mathfrak{C}$.

Information theoretic security: In a manner analogous to that of MAC schemes, we consider information theoretic security of nonce-based MAC schemes: Given a query profile $\mathfrak{C}$, $\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{NMAC}}[\lambda](\mathfrak{C})$ is the maximum of $\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{NMAC}}[\lambda](\mathcal{A})$ taken over all adversaries $\mathcal{A}$ having query profile $\mathfrak{C}$.

3 Towards Building a Variable Tag Length Nonce-Based MAC

It may appear that a variable tag length nonce-based MAC scheme can be obtained simply by truncating the output of the Wegman-Carter MAC algorithm. This, however, does not work as we show in this section. We further consider several "natural" extensions of the Wegman-Carter MAC algorithm and show that most of them are insecure. Only two of these extensions are secure: one of them is a generic construction, while we prove the security of the other in the next section. Overall, the discussion in the present section may be considered as showing the subtlety involved in constructing a variable tag length nonce-based MAC scheme.

Let $\mathcal{N}$ be the nonce space and $\mathcal{M}$ be the message space. Let $\{\mathsf{F}_K\}_{K\in\mathcal{K}}$ be a PRF such that $\mathsf{F}_K : \mathcal{N} \to \{0,1\}^n$; let $\{\mathsf{Hash}_\tau\}_{\tau\in\Theta}$ be an AXU hash function such that $\mathsf{Hash}_\tau : \mathcal{M} \to \{0,1\}^n$. Given $\{\mathsf{F}_K\}_{K\in\mathcal{K}}$ and $\{\mathsf{Hash}_\tau\}_{\tau\in\Theta}$, the Wegman-Carter MAC [26] is the following. A nonce-message pair (N, x) is mapped under a key (K, τ) to $\mathsf{F}_K(N) \oplus \mathsf{Hash}_\tau(x)$, i.e.,

$$\mathsf{WC\text{-}NMAC} : (N, x) \xrightarrow{(K,\tau)} \mathsf{F}_K(N) \oplus \mathsf{Hash}_\tau(x). \tag{6}$$

Below we argue that several natural extensions of WC-NMAC are not secure. The attacks are shown for the following specific choice of the hash function. Under a fixed representation of the elements of the finite field $\mathbb{F}_{2^n}$, we identify the elements of $\mathbb{F}_{2^n}$ with the set $\{0,1\}^n$. The specific hash function that we consider is $\mathsf{Hash}_\tau(x) = \tau x$, i.e., the output of $\mathsf{Hash}_\tau(x)$ is the n-bit string representing the product of τ and x considered as elements of $\mathbb{F}_{2^n}$. This hash function is known to be AXU. Attacks on schemes built using this specific hash function is sufficient to show that the schemes described below are not secure for an arbitrary AXU hash function. The choice of the hash function fixes the key space of the hash function to be $\Theta = \mathbb{F}_{2^n}$ and the message space $\mathcal{M}$ to be either $\mathbb{F}_{2^n}$ or $\mathbb{F}_{2^{n-8}}$, depending on the scheme.

We will use the following simple fact about the specific hash function that we consider.

Proposition 1. *Consider the AXU hash function $\{\mathsf{Hash}_\tau\}_{\tau\in\mathbb{F}_{2^n}}$ where $\mathsf{Hash}_\tau(x) = \tau x$. Let x_1 and x_2 be distinct elements of $\mathbb{F}_{2^n}$ and c be such that $\mathsf{Hash}_\tau(x_1) \oplus \mathsf{Hash}_\tau(x_2) = c$, then $\tau = c(x_1 \oplus x_2)^{-1}$.*

The most obvious approach to obtain a variable tag length scheme from (6) is to truncate the output, i.e.,

$$\mathsf{trunc} : \ (N, x, \lambda) \xrightarrow{(K,\tau)} \mathsf{msb}_\lambda(\mathsf{WC\text{-}NMAC}_{K,\tau}(N, x)) = \mathsf{msb}_\lambda(\mathsf{F}_K(N) \oplus \mathsf{Hash}_\tau(x)). \tag{7}$$

The scheme in (7) is not secure as can be seen from the following argument. Here the message space is $\mathbb{F}_{2^n}$. Let x_1, x_2 and x_3 be distinct messages and N be a nonce. The adversary makes two tag

generation queries (N, x_1, n) and $(N, x_2, n-1)$ and gets in response t_1 and t_2 respectively. So, we have the following relations: $\mathsf{F}_K(N) \oplus \mathsf{Hash}_\tau(x_1) = t_1$ and $\mathsf{msb}_{n-1}(\mathsf{F}_K(N) \oplus \mathsf{Hash}_\tau(x_2)) = t_2$. From the second relation, it follows that either $\mathsf{F}_K(N) \oplus \mathsf{Hash}_\tau(x_2) = t_2||0$ or $\mathsf{F}_K(N) \oplus \mathsf{Hash}_\tau(x_2) = t_2||1$. Using Proposition 1, the adversary solves the equations $\mathsf{Hash}_\tau(x_1) \oplus \mathsf{Hash}_\tau(x_2) = t_1 \oplus (t_2||0)$ and $\mathsf{Hash}_\tau(x_1) \oplus \mathsf{Hash}_\tau(x_2) = t_1 \oplus (t_2||1)$ for τ to obtain the solutions τ_0 and τ_1 respectively. As $\mathsf{F}_K(N) \oplus \mathsf{Hash}_\tau(x_2)$ takes exactly one of the two values $t_2||0$ or $t_2||1$, τ takes exactly one of the two values τ_0 or τ_1. Let $y_0 = t_1 \oplus \mathsf{Hash}_{\tau_0}(x_1)$. The adversary makes a verification query $(N, x_3, y_0 \oplus \mathsf{Hash}_{\tau_0}(x_3), n)$. If the verification query is successful then τ_0 is the correct value of τ. If the verification query fails, then τ_1 is the correct value of τ. Thus the adversary recovers the hash key with 2 tag generation and 1 verification queries. This shows that a simple truncation of the Wegman-Carter MAC scheme does not work.

In the scheme trunc, the output of neither F nor Hash depends on λ. To rectify this situation, one may introduce λ as part of the input of one or both of F and Hash. Another possibility is to have one or both of the keys K and τ to depend on λ. Key dependencies are achieved by using a family of independent keys $\{K_\lambda\}_{\lambda\in\mathcal{L}}$ and/or a family of independent keys $\{\tau_\lambda\}_{\lambda\in\mathcal{L}}$. The various schemes that arise from such considerations are as follows. Dependencies of input and/or key are summarised in Table 1.

$$\mathsf{NMAC\text{-}t1}_{K,\tau} : (N, x, \lambda) \xrightarrow{(K,\tau)} \mathsf{msb}_\lambda(\mathsf{F}_K(\mathsf{bin}_8(\lambda)||N) \oplus \mathsf{Hash}_\tau(x)). \tag{8}$$

$$\mathsf{NMAC\text{-}t2}_{K,\tau} : (N, x, \lambda) \xrightarrow{(K,\tau)} \mathsf{msb}_\lambda(\mathsf{F}_K(N) \oplus \mathsf{Hash}_\tau(\mathsf{bin}_8(\lambda)||x)). \tag{9}$$

$$\mathsf{NMAC\text{-}t3}_{K,\tau} : (N, x, \lambda) \xrightarrow{(K,\tau)} \mathsf{msb}_\lambda(\mathsf{F}_K(\mathsf{bin}_8(\lambda)||N) \oplus \mathsf{Hash}_\tau(\mathsf{bin}_8(\lambda)||x)). \tag{10}$$

$$\mathsf{NMAC\text{-}Generic}_{(K_\lambda,\tau_\lambda)_{\lambda\in\mathcal{L}}} : (N, x, \lambda) \xrightarrow{(K_\lambda,\tau_\lambda)} \mathsf{msb}_\lambda(\mathsf{F}_{K_\lambda}(N) \oplus \mathsf{Hash}_{\tau_\lambda}(x)). \tag{11}$$

$$\mathsf{NMAC\text{-}t4}_{(K_\lambda,\tau)_{\lambda\in\mathcal{L}}} : (N, x, \lambda) \xrightarrow{(K_\lambda,\tau)} \mathsf{msb}_\lambda(\mathsf{F}_{K_\lambda}(N) \oplus \mathsf{Hash}_\tau(x)). \tag{12}$$

$$\mathsf{NMAC\text{-}t5}_{(K_\lambda,\tau)_{\lambda\in\mathcal{L}}} : (N, x, \lambda) \xrightarrow{(K_\lambda,\tau)} \mathsf{msb}_\lambda(\mathsf{F}_{K_\lambda}(N) \oplus \mathsf{Hash}_\tau(\mathsf{bin}_8(\lambda)||x)). \tag{13}$$

$$\mathsf{NMAC\text{-}t6}_{(K,\tau_\lambda)_{\lambda\in\mathcal{L}}} : (N, x, \lambda) \xrightarrow{(K,\tau_\lambda)} \mathsf{msb}_\lambda(\mathsf{F}_K(N) \oplus \mathsf{Hash}_{\tau_\lambda}(x)). \tag{14}$$

$$\mathsf{NMAC}_{(K,\tau_\lambda)_{\lambda\in\mathcal{L}}} : (N, x, \lambda) \xrightarrow{(K,\tau_\lambda)} \mathsf{msb}_\lambda(\mathsf{F}_K(\mathsf{bin}_8(\lambda)||N) \oplus \mathsf{Hash}_{\tau_\lambda}(x)). \tag{15}$$

Nonce stealing: Finney [11] had suggested that the nonce may be reduced by a few bits and a binary encoding of the tag length be inserted. In the present context, this refers to letting the input of F depend on the tag length. From Table 1, we see that the schemes NMAC-t1, NMAC-t3 and NMAC use nonce stealing. While NMAC is secure (as proved later), schemes NMAC-t1 and NMAC-t3 are insecure. So, nonce stealing by itself does not guarantee security.

For the ensuing discussion, we will consider the message space for the schemes NMAC-t1, NMAC-Generic, NMAC-t4 and NMAC-t6 to be $\mathbb{F}_{2^n}$, and that for NMAC-t2, NMAC-t3 and NMAC-t5 to be $\mathbb{F}_{2^{n-8}}$. The scheme NMAC is discussed in more generality in the next section.

The schemes NMAC-t1, NMAC-t2 and NMAC-t3 are based only on input dependencies. In NMAC-t1, the input to F depends on the tag length but, the input to Hash does not; in NMAC-t2, the input to Hash depends on the tag length but, the input to F does not; and in NMAC-t3, the inputs to both F and Hash depend on the tag length. None of these schemes are secure.

1. Algorithm 1 presents an attack on NMAC-t1. In the attack, the tag generation and verification oracles are denoted by $\mathcal{O}_g$ and $\mathcal{O}_v$ respectively. The intuition behind the attack is the

Table 1: For the schemes in (8) to (15), a summary of whether the input and/or the key of F and/or Hash depend on the tag length λ.

scheme	F		Hash		secure?
	i/p	key	i/p	key	
NMAC-t1	yes	no	no	no	no
NMAC-t2	no	no	yes	no	no
NMAC-t3	yes	no	yes	no	no
NMAC-Generic	no	yes	no	yes	yes
NMAC-t4	no	yes	no	no	no
NMAC-t5	no	yes	yes	no	no
NMAC-t6	no	no	no	yes	no
NMAC	yes	no	no	yes	yes

following. The key (K, τ) of the scheme does not depend on λ. In particular, as the hash key τ does not depend on λ, the attack retrieves τ using a smaller value of λ and uses it for the forgery with the target λ successfully. Retrieving τ using a smaller value of λ requires significantly lesser number of oracle queries than that required for an attack by exhaustive search for the target λ. The analysis of the attack is given in Proposition 2.

2. For NMAC-t2, the attack on NMAC-t1 works as it is. The attack can, however, be modified to a more efficient one in the following manner. Instead of the verification queries inside the do-while loop in Steps 7 to 11, the adversary can make another tag generation query on the tuple (N_1, x_2, λ) to get $\mathsf{tag}^{(2)}$, so that,

$$\mathsf{msb}_\lambda(\mathsf{F}_K(N_1) \oplus \mathsf{Hash}_\tau(\mathsf{bin}_8(\lambda)||x_2)) = \mathsf{tag}^{(2)}. \tag{16}$$

The adversary takes $\mathsf{msb}_{\lambda_1}(\cdot)$ of (16) and gets τ_c as in the attack in Algorithm 1.

Now the adversary *forges* with $(N_1, x, \mathsf{Hash}_{\tau_c}(\mathsf{bin}_8(\lambda)||x_2) \oplus \mathsf{tag}^{(2)} \oplus \mathsf{Hash}_{\tau_c}(\mathsf{bin}_8(\lambda)||x),\ \lambda)$. So, here the adversary makes 2 tag generation queries and at most $2^{n-\lambda_1} + 1$ verification queries including the forgery.

3. For NMAC-t3 the attack on NMAC-t1 works as it is with the forgery $(N_1, x, \mathsf{Hash}_{\tau_c}(\mathsf{bin}_8(\lambda)||x) \oplus \mathsf{Hash}_{\tau_c}(\mathsf{bin}_8(\lambda)||x_4) \oplus \mathsf{tag}^{(4)}, \lambda)$. Here also, the adversary makes 2 tag generation queries and at most $2^{\lambda_1} + 2^{n-\lambda_1} + 1$ verification queries including the forgery.

Proposition 2. *The attack given in Algorithm 1 on the scheme* NMAC-t1 *given in (8) forges with probability 1 and requires 2 tag generation queries and at most* $2^{\lambda_1} + 2^{n-\lambda_1} + 1$ *verification queries including the forgery.*

Proof. That the attack mentioned in Algorithm 1 forges with probability 1 is proved if it can be shown that the forgery returned by the attack in Step 25 is accepted, i.e. the corresponding response from $\mathcal{O}_v$ is true.
From Step 4 we get,

$$\mathsf{msb}_{\lambda_1}(\mathsf{F}_K(\mathsf{bin}_8(\lambda_1)||N_1) \oplus \mathsf{Hash}_\tau(x_1)) = \mathsf{tag}^{(1)}. \tag{17}$$

Algorithm 1 Attack on NMAC-t1 for $\lambda = n$.

set $\lambda \leftarrow n$;
choose $\lambda_1 \in \mathcal{L}$, such that $\lambda_1 < \lambda$;
choose any $N_1 \in \mathcal{N}$ and any $x_1 \in \mathcal{M}$ and let
  $\mathsf{tag}^{(1)} \leftarrow \mathcal{O}_g(N_1, x_1, \lambda_1)$;
set $\mathcal{D} \leftarrow \{\}$;
choose $x_2 \in \mathcal{M}$, such that $x_2 \neq x_1$;
do
    choose $\mathsf{tag}^{(2)} \leftarrow \{0,1\}^{\lambda_1} \setminus \mathcal{D}$;
    set $\mathcal{D} \leftarrow \mathcal{D} \cup \mathsf{tag}^{(2)}$;
    $\mathcal{R}_v^{(2)} \leftarrow \mathcal{O}_v(N_1, x_2, \mathsf{tag}^{(2)}, \lambda_1)$;
while $\mathcal{R}_v^{(2)} = \mathsf{false}$;
set $\mathcal{C} \leftarrow \{\}$;
choose $x_3 \leftarrow \mathcal{M} \setminus \{x_1, x_2\}$;
do
    choose $c \leftarrow \{0,1\}^{n-\lambda_1} \setminus \mathcal{C}$;
    set $\mathcal{C} \leftarrow \mathcal{C} \cup c$;
    using Proposition 1 solve $\mathsf{Hash}_\tau(x_1) \oplus \mathsf{Hash}_\tau(x_2) = (\mathsf{tag}^{(1)} \oplus \mathsf{tag}^{(2)})||c$
      for τ and let the solution be τ_c;
    set $x_c \leftarrow \mathsf{tag}^{(1)} \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_c}(x_1))$;
    $\mathcal{R}_v^{(3)} \leftarrow \mathcal{O}_v(N_1, x_3, x_c \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_c}(x_3)), \lambda_1)$;
while $\mathcal{R}_v^{(3)} = \mathsf{false}$;
choose any $x_4 \in \mathcal{M}$ and let
  $\mathsf{tag}^{(4)} \leftarrow \mathcal{O}_g(N_1, x_4, \lambda)$;
choose any $x \in \mathcal{M} \setminus \{x_4\}$ and
  return $(N_1, x, \mathsf{Hash}_{\tau_c}(x) \oplus \mathsf{Hash}_{\tau_c}(x_4) \oplus \mathsf{tag}^{(4)}, \lambda)$.

As $\mathsf{NMAC\text{-}t1}(N_1, x_2, \lambda_1) \in \{0,1\}^{\lambda_1}$ and $\mathsf{tag}^{(2)}$ is varied through $\{0,1\}^{\lambda_1}$, the do-while loop in Steps 7 to 11 terminates and for the terminating query, i.e. for the query for which $\mathcal{R}_v^{(2)} = \mathsf{true}$, we get,

$$\mathsf{msb}_{\lambda_1}(\mathsf{F}_K(\mathsf{bin}_8(\lambda_1)||N_1) \oplus \mathsf{Hash}_\tau(x_2)) = \mathsf{tag}^{(2)}. \tag{18}$$

So, from (17) and (18) we get,

$$\mathsf{msb}_{\lambda_1}(\mathsf{Hash}_\tau(x_1) \oplus \mathsf{Hash}_\tau(x_2)) = \mathsf{tag}^{(1)} \oplus \mathsf{tag}^{(2)}. \tag{19}$$

Here $\mathsf{tag}^{(1)} \oplus \mathsf{tag}^{(2)}$ is a λ_1-bit binary string. Following Proposition 1, for each choice of c in the do-while loop in Steps 14 to 21, the equation in Step 17 can be solved to get τ_c and x_c.
The fact that $\mathsf{Hash}_\tau(x_1) \oplus \mathsf{Hash}_\tau(x_2) \in \{0,1\}^n$ and (19) suggest that there is a correct c, such that the equation in Step 17 holds and we consider that iteration of the do-while loop which deals with this particular c. The τ_c obtained in this iteration is the actual hash key used in the scheme. So,

$$\begin{aligned}
&\mathsf{NMAC\text{-}t1}(N_1, x_3, \lambda_1) \\
&= \mathsf{msb}_{\lambda_1}(\mathsf{F}_K(\mathsf{bin}_8(\lambda_1)||N_1) \oplus \mathsf{Hash}_{\tau_c}(x_3)) \\
&= \mathsf{tag}^{(1)} \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_c}(x_1)) \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_c}(x_3)) \qquad (20) \\
&= x_c \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_c}(x_3)). \qquad (21)
\end{aligned}$$

The expression in (20) comes from (17) and that in (21) comes from Step 19 in Algorithm 1. Hence, in this particular iteration of the do-while loop, $\mathcal{R}_v^{(3)} = \mathsf{true}$ and the loop terminates.
Next, from Step 23, we get

$$\mathsf{msb}_{\lambda}(\mathsf{F}_K(\mathsf{bin}_8(\lambda)||N_1) \oplus \mathsf{Hash}_{\tau_c}(x_4)) = \mathsf{tag}^{(4)}. \tag{22}$$

Hence,

$$\mathsf{F}_K(\mathsf{bin}_8(\lambda)||N_1) = \mathsf{Hash}_{\tau_c}(x_4) \oplus \mathsf{tag}^{(4)}; \tag{23}$$

msb_λ is omitted as $\lambda = n$.
Now, for the $x \in \mathcal{M} \setminus \{x_4\}$,

$$\begin{aligned}
&\mathsf{NMAC\text{-}t1}(N_1, x, \lambda) \\
&= \mathsf{msb}_{\lambda}(\mathsf{F}_K(\mathsf{bin}_8(\lambda)||N_1) \oplus \mathsf{Hash}_{\tau_c}(x)) \\
&= \mathsf{Hash}_{\tau_c}(x_4) \oplus \mathsf{tag}^{(4)} \oplus \mathsf{Hash}_{\tau_c}(x), \qquad (24)
\end{aligned}$$

which is returned as the tag for (N_1, x, λ) in the forgery and hence, the corresponding response from $\mathcal{O}_v$ is true with probability 1, which proves the first part of the result.

In the attack, there are 2 tag generation queries in Steps 4 and 23. In each iteration of each of the do-while loops in Steps 7 to 11 and 14 to 21, one verification query is made. The do-while loop in Steps 7 to 11 iterates at most 2^{λ_1} times for different values of $\mathsf{tag}^{(2)}$ and that in Steps 14 to 21 iterates at most $2^{n-\lambda_1}$ times for different values of c. The forgery returned in Step 25 is another verification query. Hence, the attack requires 2 tag generation queries and at most $2^{\lambda_1} + 2^{n-\lambda_1} + 1$ verification queries including the forgery. □

Next we consider allowing the keys to depend on λ. The generic scheme of this type is $\mathsf{NMAC\text{-}Generic}$. This scheme can be considered to be a collection of $\#\mathcal{L}$ independent $\mathsf{WC\text{-}NMAC}$ schemes, one for each value of λ. Each of the individual schemes for fixed values of λ are already known to be secure. Since the keys of the various schemes are independent, it can be argued that

the collection is also secure. The problem, however, is that size of the key increases by a factor of $\#\mathcal{L}$. So, NMAC-Generic cannot be considered to be a practical solution to the problem of obtaining a variable tag length nonce-based MAC scheme.

The remaining schemes consider various possibilities for reducing the key size.

1. The scheme NMAC-t4 uses independent K_λ and the same τ for all λ. For this scheme, the attack for the scheme NMAC-t1 works with the forgery $(N, x, \mathsf{Hash}_{\tau_c}(x) \oplus \mathsf{Hash}_{\tau_c}(x_4) \oplus \mathsf{tag}^{(4)}, \lambda)$; the adversary makes 2 tag generation queries and at most $2^{\lambda_1}+2^{n-\lambda_1}+1$ verification queries including the forgery.

2. The scheme NMAC-t5 uses independent K_λ and the same τ for all λ. It introduces λ into the input of Hash. This, however, is not sufficient. The attack on the scheme NMAC-t1 works with the forgery $(N, x, \mathsf{Hash}_{\tau_c}(\mathsf{bin}_8(\lambda)||x) \oplus \mathsf{Hash}_{\tau_c}(\mathsf{bin}_8(\lambda)||x_4) \oplus \mathsf{tag}^{(4)}, \lambda)$; the adversary makes 2 tag generation queries and at most $2^{\lambda_1} + 2^{n-\lambda_1} + 1$ verification queries including the forgery.

3. The scheme NMAC-t6 uses independent τ_λ and the same key K for all λ and the input of F does not depend on λ. The insecurity of NMAC-t6 is discussed in Appendix A.

4. The final scheme NMAC uses independent τ_λ and the same key K for all λ. In this case, the input of F depends on λ. This scheme is secure. The information theoretic analysis of the security of this scheme is provided in the next section.

4 Secure and Efficient Nonce-Based MAC Schemes with Variable Length Tag

We start with the scheme NMAC given in (15). We carry out an information theoretic analysis of this scheme. To this end, we consider the scheme obtained by replacing F_K with a random function $f : \{0,1\}^n \to \{0,1\}^n$. The tag generation algorithm for this scheme is shown in Table 2. We require a hash family $\{\mathsf{Hash}_\tau\}_{\tau\in\Theta}$, where for each $\tau \in \Theta$, $\mathsf{Hash}_\tau : \mathcal{M} \to \{0,1\}^n$, with $\mathcal{M} = \cup_{i=0}^{L}\{0,1\}^i$ for some sufficiently large positive integer L.

The nonce space for the scheme NMAC is $\mathcal{N} = \{0,1\}^{n-8}$ and the message space is $\mathcal{M}$. Let $\mathcal{L} \subseteq \{1, \ldots, \min(255, n)\}$ be the allowed set of tag lengths. So, tag lengths can be represented by bytes. The key space for NMAC is $\Theta^{\#\mathcal{L}}$, i.e., a particular key is a tuple $(\tau_\lambda)_{\lambda\in\mathcal{L}}$. The key generation algorithm consists of choosing τ_λ independently and uniformly at random from Θ for each λ. The verification algorithm is as follows. Given $(N, x, \mathsf{tag}, \lambda)$, compute $\mathsf{tag}' = \mathsf{NMAC.Gen}_{(\tau_\lambda)_{\lambda\in\mathcal{L}}}(N, x, \lambda)$; if $\mathsf{tag} = \mathsf{tag}'$ then return true, else return false.

Here f is a random function but, not necessarily a uniform random function. Given q pairs $(a_1, b_1), \ldots, (a_q, b_q)$, the q-interpolation probability [5] of f is defined to be $\Pr[f(a_1) = b_1, \ldots, f(a_q) = b_q]$. Following the analysis in [5], the security bound for the resulting scheme is obtained in terms of the interpolation probability of f. Known bounds on the interpolation probability of uniform random function and uniform random permutation provide the corresponding bounds on the security of the resulting NMAC schemes.

Theorem 1. *In* NMAC *defined in Table 2, suppose that the hash function* $\{\mathsf{Hash}_\tau\}_{\tau\in\Theta}$ *is* ε*-AXU, where* $\varepsilon(\ell, \ell') \geq 1/2^n$ *for all* $\ell, \ell' \leq L$.

Fix a query profile $\mathfrak{C}$. *For* $\lambda \in \mathcal{L}$, *let* $q_{g,\lambda}$ *(resp.* $q_{v,\lambda}$*) be the number of tag generation (resp. verification) queries for* λ *which are in* $\mathfrak{C}$. *Let* λ *be such that* $q_{v,\lambda} \geq 1$ *and for* $1 \leq i \leq q_{v,\lambda}$, *let* $Q_{v,\lambda}^{(i)} = (N_{v,\lambda}^{(i)}, x_{v,\lambda}^{(i)}, \mathsf{tag}_{v,\lambda}^{(i)}, \lambda)$ *be the* i*-th verification query with tag length* λ. *Let* $\ell_{v,\lambda}^{(i)} = \mathsf{len}(x_{v,\lambda}^{(i)})$.

Table 2: A secure and efficient NMAC scheme from a random function.

$\mathsf{NMAC.Gen}_{(\tau_\lambda)_{\lambda\in\mathcal{L}}}(N, x, \lambda)$
  $Q = f(\mathsf{bin}_8(\lambda)||N)$;
  $R = Q \oplus \mathsf{Hash}_{\tau_\lambda}(x)$;
  $\mathsf{tag} = \mathsf{msb}_\lambda(R)$;
return tag.

Corresponding to $Q^{(i)}_{v,\lambda}$, there is at most one tag generation query $Q^{(i^\star)}_{g,\lambda} = (N^{(i^\star)}_{g,\lambda}, x^{(i^\star)}_{g,\lambda}, \lambda)$ such that $N^{(i)}_{v,\lambda} = N^{(i^\star)}_{g,\lambda}$. Let $\ell^{(i^\star)}_{g,\lambda} = \mathsf{len}(x^{(i^\star)}_{g,\lambda})$ if there is such a $Q^{(i^\star)}_{g,\lambda}$, otherwise $\ell^{(i^\star)}_{g,\lambda}$ is undefined.

Fix $\lambda_0 \in \mathcal{L}$. Let $\mathcal{S}_{\lambda_0}$ be the set of all queries made by the adversary other than the verification queries for tag length λ_0. Suppose that the queries in $\mathcal{S}_{\lambda_0}$ give rise to at most q distinct (nonce, tag-length) values. Further, suppose that the i-interpolation probability of f is at most $\delta_i/(2^n)^i$.

Then

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{NMAC}}[\lambda_0](t, \mathfrak{C}) \;\leq\; \frac{1}{2^{\lambda_0}} \times \sum_{1\leq i\leq q_{v,\lambda_0}} \gamma_i \tag{25}$$

where $\gamma_i = 2^n\delta_q\varepsilon\left(\ell^{(i)}_{v,\lambda_0}, \ell^{(i^\star)}_{g,\lambda_0}\right)$ if there is a $Q^{(i^\star)}_{g,\lambda_0}$ corresponding to $Q^{(i)}_{v,\lambda_0}$ with $N^{(i)}_{v,\lambda_0} = N^{(i^\star)}_{g,\lambda_0}$; otherwise $\gamma_i = \delta_{q+1}$.

Remark: It has been proved in [5], that for $1 \leq j \leq 2^n$, if f is a uniform random function, then $\delta_j = 1$, and if f is a uniform random permutation, then $\delta_j \leq (1 - (j-1)/2^n)^{-j/2}$.

Proof. The proof builds upon and generalises ideas used in the security proof of the Wegman-Carter nonce-based MAC scheme given in [5].

Let $\mathcal{A}$ be an adversary attacking the authenticity of NMAC. The result concerns information theoretic security and so we consider the adversary to be deterministic. $\mathcal{A}$ makes a number of queries to its oracles and receives the appropriate responses. The interaction of $\mathcal{A}$ with its two oracles is given by a transcript $\mathcal{T}$ which is a list of the queries made by $\mathcal{A}$ and the responses it received in return. The adversary's view of the oracles is completely determined by the transcript $\mathcal{T}$. By $\mathcal{A}(\mathcal{T})$, we will denote the interaction of $\mathcal{A}$ with the oracles as given by the transcript $\mathcal{T}$. The responses to the queries made by $\mathcal{A}$ are computed using the random function f and hence are random variables. Since $\mathcal{A}$ is deterministic, the randomness in a transcript $\mathcal{T}$ arises only from these responses. By $\mathsf{succ}(\mathcal{A}(\mathcal{T}), \lambda_0)$ we will denote the event that the adversary $\mathcal{A}$ with transcript $\mathcal{T}$ makes a verification query for tag length λ_0 which returns true. So, if the transcript $\mathcal{T}$ corresponds to the query profile $\mathfrak{C}$, then $\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{NMAC}}[\lambda_0](t, \mathfrak{C}) = \Pr[\mathsf{succ}(\mathcal{A}(\mathcal{T}), \lambda_0)]$.

The first reduction is to assume that $q_{v,\lambda_0} = 1$. If $q_{v,\lambda_0} = 0$, i.e., $\mathcal{A}$ does not make any verification query, then clearly, $\mathcal{A}$ has advantage 0 so that the theorem is trivially proved. So, suppose that $\mathcal{A}$ with transcript $\mathcal{T}$ makes $q_{v,\lambda_0} > 1$ verification queries for tag-length λ_0. Let $\mathcal{E}$ be the event that the first verification query for the tag length λ_0 is successful and $\mathcal{S}$ be the event that one of the later verification queries for the tag length λ_0 is successful. So,

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{NMAC}}[\lambda_0](\mathcal{A}) = \Pr[\mathsf{succ}(\mathcal{A}(\mathcal{T}), \lambda_0)] = \Pr[\mathcal{E} \vee \mathcal{S}] = \Pr[\mathcal{E} \vee (\overline{\mathcal{E}} \wedge \mathcal{S})] \leq \Pr[\mathcal{E}] + \Pr[\overline{\mathcal{E}} \wedge \mathcal{S}].$$

Let $\mathcal{A}'$ be an adversary with a transcript $\mathcal{T}'$ which is obtained from $\mathcal{T}$ by discarding all queries after the first verification query for tag length λ_0. Let $\mathcal{A}''$ be an adversary with a transcript $\mathcal{T}''$ obtained

from $\mathcal{T}$ in the following manner: the first verification query for tag length λ_0 is dropped from $\mathcal{T}$; instead $\mathcal{A}''$ takes the answer false as the response to this query. (Note that since we are disallowing useless queries, there could not have been a previous tag generation query for tag length λ_0 with the same nonce and message as that of the first verification query for tag length λ_0.) We have $\Pr[\mathsf{succ}(\mathcal{A}'(\mathcal{T}'), \lambda_0)] = \Pr[\mathcal{E}]$ and $\Pr[\mathsf{succ}(\mathcal{A}''(\mathcal{T}''), \lambda_0)] = \Pr[\overline{\mathcal{E}} \wedge \mathcal{S}]$. Note that $\mathcal{A}''$ makes $q_{v,\lambda_0} - 1$ verification queries for tag length λ_0. So, the problem of proving the result for q_{v,λ_0} verification queries has been reduced to the problem of proving the result for $q_{v,\lambda_0} - 1$ verification queries. Proceeding by induction, to prove the bound given in (25), it is sufficient to consider an adversary which makes exactly one verification query for tag length λ_0. Let the single verification query for tag length λ_0 be $(N, x, \mathsf{tag}, \lambda_0)$.

The second reduction is to ignore all queries in $\mathcal{T}$ after the verification query for tag length λ_0. Such queries have no effect on the success probability of the verification query for tag length λ_0.

The third reduction is the following. If the queries in $\mathcal{S}_{\lambda_0}$ give rise to less than q distinct (nonce, tag-length) values, then insert additional tag generation queries to the transcript with (nonce, tag-length) values not equal to (N, λ_0) such that the queries in the augmented $\mathcal{S}_{\lambda_0}$ give rise to exactly q distinct (nonce, tag-length) values. Such augmentation of the transcript does not decrease the adversary's advantage.

In view of the above reductions, it is sufficient to consider an adversary $\mathcal{A}$ with a transcript $\mathcal{T}$ where the last query is the verification query $(N, x, \mathsf{tag}, \lambda_0)$ for tag length λ_0 and the queries in $\mathcal{S}_{\lambda_0}$ give rise to exactly q distinct (nonce, tag-length) values. The transcript $\mathcal{T}$ can contain any number of tag generation queries for the tag length λ_0. However, by the restriction that among the tag generation queries, the (nonce, tag-length) pair cannot repeat, $\mathcal{T}$ can contain at most one tag generation query of the form (N, x', λ_0). For $\lambda \neq \lambda_0$, the transcript $\mathcal{T}$ can contain multiple verification queries with the same value for the (nonce, λ) pair. So, the total number of queries in $\mathcal{S}_{\lambda_0}$ can be greater than q.

Let $\mathfrak{N} = \mathsf{bin}_8(\lambda_0)||N$, $Q = f(\mathfrak{N})$ and $\tau_0 = \tau_{\lambda_0}$. Let the q distinct values of (nonce, tag-length) pairs arising from the queries in $\mathcal{S}_{\lambda_0}$ be $(N^{(1)}, \lambda^{(1)}), \ldots, (N^{(q)}, \lambda^{(q)})$. For $i = 1, \ldots, q$, let $\mathfrak{N}^{(i)} = \mathsf{bin}_8(\lambda^{(i)})||N^{(i)}$ and $Q^{(i)} = f(\mathfrak{N}^{(i)})$. Define $\mathbf{Q} = (Q^{(1)}, \ldots, Q^{(q)})$. Let q' be the number of distinct tag-length values arising from the queries in $\mathcal{S}_{\lambda_0}$ and let $\lambda^{(1)}, \ldots, \lambda^{(q')}$ be these tag lengths. For $i = 1, \ldots, q'$, define $\tau_i = \tau_{\lambda^{(i)}}$ and $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_{q'})$. The entire randomness in the transcript arises from $\mathbf{Q}$ and $\boldsymbol{\tau}$.

Consider the final verification query $(N, x, \mathsf{tag}, \lambda_0)$ and let $\ell = \mathsf{len}(x)$. Let $\ell^{(\star)} = \mathsf{len}(x^{(\star)})$ if there is a prior tag generation query $(N^{(\star)}, x^{(\star)}, \lambda^{(\star)})$ (with response $\mathsf{tag}^{(\star)}$) such that $N^{(\star)} = N$ and $\lambda^{(\star)} = \lambda_0$; otherwise, $\ell^{(\star)}$ is undefined. Let $\gamma = 2^n \delta_q \varepsilon(\ell, \ell^{(\star)})$ if $\ell^{(\star)}$ is defined, otherwise, $\gamma = \delta_{q+1}$. To prove the theorem, it is sufficient to show

$$\Pr[\mathsf{succ}(\mathcal{A}(\mathcal{T}), \lambda_0)] \;\leq\; \gamma / 2^{\lambda_0}. \tag{26}$$

The verification query is successful if $\mathsf{tag} = \mathsf{msb}_{\lambda_0}(Q \oplus \mathsf{Hash}_{\tau_0}(x))$. So,

$$\Pr[\mathsf{succ}(\mathcal{A}(\mathcal{T}), \lambda_0)] \;=\; \Pr[\mathsf{msb}_{\lambda_0}(Q \oplus \mathsf{Hash}_{\tau_0}(x)) = \mathsf{tag}]. \tag{27}$$

We consider the probability on the right hand side of (27) under two cases.

The first case is when there is no tag generation query having (nonce, tag-length) pair to be equal to (N, λ_0) in $\mathcal{T}$. In this case, $\mathfrak{N}^{(1)}, \ldots, \mathfrak{N}^{(q)}, \mathfrak{N}$ are distinct values to which f is applied. Since the adversary is adaptive, the x and tag in the final verification query are functions of the earlier responses it received and in turn are functions of $\mathbf{Q}$ and $\boldsymbol{\tau}$. We write $x \equiv x(\mathbf{Q}, \boldsymbol{\tau})$ and

$\mathsf{tag} \equiv \mathsf{tag}(\mathbf{Q}, \boldsymbol{\tau})$ to emphasise this functional dependence. Let a and $\mathbf{a}$ be arbitrary values of τ_0 and $\boldsymbol{\tau}$. Let $b_1, \ldots, b_q$ be arbitrary n-bit strings and let $\mathbf{b} = (b^{(1)}, \ldots, b^{(q)})$. So,

$$\begin{aligned}
&\Pr[\mathsf{msb}_{\lambda_0}(Q \oplus \mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \boldsymbol{\tau}))) = \mathsf{tag}(\mathbf{Q}, \boldsymbol{\tau})] \\
&= \Pr[\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{Q}, \boldsymbol{\tau}) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \boldsymbol{\tau})))] \\
&= \sum_{\mathbf{a}, a} \Pr[\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{Q}, \boldsymbol{\tau}) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \boldsymbol{\tau}))) \wedge (\boldsymbol{\tau} = \mathbf{a}) \wedge (\tau_0 = a)] \\
&= \sum_{\mathbf{a}, a} \Pr[\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{Q}, \mathbf{a}) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{a}(x(\mathbf{Q}, \mathbf{a}))) \wedge (\boldsymbol{\tau} = \mathbf{a}) \wedge (\tau_0 = a)] \\
&= \sum_{\mathbf{a}, a} \Pr[\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{Q}, \mathbf{a}) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{a}(x(\mathbf{Q}, \mathbf{a})))] \Pr[(\boldsymbol{\tau} = \mathbf{a}) \wedge (\tau_0 = a)].
\end{aligned} \tag{28}$$

Let c be an arbitrary $(n - \lambda_0)$-bit binary string. We consider

$$\begin{aligned}
&\Pr[\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{Q}, \mathbf{a}) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{a}(x(\mathbf{Q}, \mathbf{a})))] \\
&= \sum_{\mathbf{b}} \Pr[\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{Q}, \mathbf{a}) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{a}(x(\mathbf{Q}, \mathbf{a}))) \wedge (\mathbf{Q} = \mathbf{b})] \\
&= \sum_{\mathbf{b}} \Pr[\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{b}, \mathbf{a}) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{a}(x(\mathbf{b}, \mathbf{a}))) \wedge (\mathbf{Q} = \mathbf{b})] \\
&= \sum_{\mathbf{b}} \Pr[\mathsf{msb}_{\lambda_0}(Q) = b \wedge (\mathbf{Q} = \mathbf{b})] \quad \text{where } b = \mathsf{tag}(\mathbf{b}, \mathbf{a}) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{a}(x(\mathbf{b}, \mathbf{a}))) \\
&= \sum_{\mathbf{b}} \Pr[\mathsf{msb}_{\lambda_0}(f(\mathfrak{N})) = b, f(\mathfrak{N}^{(1)}) = b^{(1)}, \ldots, f(\mathfrak{N}^{(q)}) = b^{(q)}] \\
&= \sum_{\mathbf{b}} \sum_{c} \Pr[f(\mathfrak{N}) = b || c, f(\mathfrak{N}^{(1)}) = b^{(1)}, \ldots, f(\mathfrak{N}^{(q)}) = b^{(q)}] \\
&\leq \sum_{\mathbf{b}} 2^{n-\lambda_0} \delta_{q+1} / (2^n)^{q+1} \\
&= 2^{n-\lambda_0} \delta_{q+1} / 2^n = \gamma / 2^{\lambda_0}.
\end{aligned} \tag{29}$$

Combining (28) and (29), we have

$$\begin{aligned}
&\Pr[\mathsf{msb}_{\lambda_0}(Q \oplus \mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \boldsymbol{\tau}))) = \mathsf{tag}(\mathbf{Q}, \boldsymbol{\tau})] \\
&= \sum_{\mathbf{a}, a} \Pr[\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{Q}, \mathbf{a}) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{a}(x(\mathbf{Q}, \mathbf{a})))] \Pr[(\boldsymbol{\tau} = \mathbf{a}) \wedge (\tau_0 = a)] \\
&\leq \gamma / 2^{\lambda_0} \sum_{\mathbf{a}, a} \Pr[(\boldsymbol{\tau} = \mathbf{a}) \wedge (\tau_0 = a)] = \gamma / 2^{\lambda_0}.
\end{aligned} \tag{30}$$

This proves the first case.
In the second case, let the transcript $\mathcal{T}$ be such that there is a tag generation query $(N^{(\star)}, x^{(\star)}, \lambda^{(\star)})$ (with response $\mathsf{tag}^{(\star)}$) where $N^{(\star)} = N$ and $\lambda^{(\star)} = \lambda_0$. Note that by the query restriction on the adversary, $x^{(\star)} \neq x$. Let $\mathfrak{N}^{(\star)} = \mathsf{bin}_8(\lambda^{(\star)}) || N^{(\star)}$, $Q^{(\star)} = f(\mathfrak{N}^{(\star)})$ and $\tau_\star = \tau_{\lambda^{(\star)}}$. Then $Q^{(\star)} = Q$ and $\tau_\star = \tau_0$. Let $\mathbf{Q}$ be the vector consisting of $Q^{(1)}, \ldots, Q^{(q)}$ but, not containing $Q^{(\star)}$ and let $\boldsymbol{\tau}$ be the vector consisting of $\tau_1, \ldots, \tau_{q'}$ but, not containing $\tau_\star$. So, $\mathbf{Q}$ is a vector having $q-1$ components and $\boldsymbol{\tau}$ is a vector having $q'-1$ components. In this case, $x \equiv x(\mathbf{Q}, \boldsymbol{\tau}, \mathsf{tag}^{(\star)})$ and $\mathsf{tag} \equiv \mathsf{tag}(\mathbf{Q}, \boldsymbol{\tau}, \mathsf{tag}^{(\star)})$.

Due to the adaptive nature of the adversary, $x^{(\star)}$ is also a function of portions of $\mathbf{Q}$ and $\boldsymbol{\tau}$ which corresponds to the queries earlier to $(N^{(\star)}, x^{(\star)}, \lambda^{(\star)})$. Hence, we write $x^{(\star)} \equiv x^{(\star)}(\mathbf{Q}, \boldsymbol{\tau})$. Note that τ_0 is independent of $\boldsymbol{\tau}$.

Let $\mathbf{a}$ and t be arbitrary values for $\boldsymbol{\tau}$ and $\mathsf{tag}^{(\star)}$ respectively. Then

$$\begin{aligned}
&\Pr[\mathsf{msb}_{\lambda_0}(Q \oplus \mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \boldsymbol{\tau}, \mathsf{tag}^{(\star)}))) = \mathsf{tag}(\mathbf{Q}, \boldsymbol{\tau}, \mathsf{tag}^{(\star)})] \\
&= \sum_{\mathbf{a},t} \Pr[(\mathsf{msb}_{\lambda_0}(Q \oplus \mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \boldsymbol{\tau}, \mathsf{tag}^{(\star)}))) = \mathsf{tag}(\mathbf{Q}, \boldsymbol{\tau}, \mathsf{tag}^{(\star)})) \wedge (\boldsymbol{\tau} = \mathbf{a}) \\
&\qquad \wedge (\mathsf{tag}^{(\star)} = t)] \\
&= \sum_{\mathbf{a},t} \Pr[(\mathsf{msb}_{\lambda_0}(Q \oplus \mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \mathbf{a}, t))) = \mathsf{tag}(\mathbf{Q}, \mathbf{a}, t)) \\
&\qquad \wedge (\mathsf{msb}_{\lambda_0}(Q \oplus \mathsf{Hash}_{\tau_0}(x^{(\star)}(\mathbf{Q}, \mathbf{a}))) = t) \wedge (\boldsymbol{\tau} = \mathbf{a})] \\
&= \sum_{\mathbf{a}} \Big(\sum_{t} \Pr[(\mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \mathbf{a}, t)) \oplus \mathsf{Hash}_{\tau_0}(x^{(\star)}(\mathbf{Q}, \mathbf{a}))) = \mathsf{tag}(\mathbf{Q}, \mathbf{a}, t) \oplus t) \\
&\qquad \wedge (\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{Q}, \mathbf{a}, t) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \mathbf{a}, t))))] \Big) \times \Pr[\boldsymbol{\tau} = \mathbf{a}]. \qquad (31)
\end{aligned}$$

Let $\mathbf{b}$ and a be an arbitrary value of $\mathbf{Q}$ and τ_0. Let c_1 and c_2 be arbitrary $(n - \lambda_0)$-bit strings. We consider

$$\begin{aligned}
&\Pr[(\mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \mathbf{a}, t)) \oplus \mathsf{Hash}_{\tau_0}(x^{(\star)}(\mathbf{Q}, \mathbf{a}))) = \mathsf{tag}(\mathbf{Q}, \mathbf{a}, t) \oplus t) \\
&\qquad \wedge (\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{Q}, \mathbf{a}, t) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \mathbf{a}, t))))] \\
&= \sum_{\mathbf{b}} \Pr[(\mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \mathbf{a}, t)) \oplus \mathsf{Hash}_{\tau_0}(x^{(\star)}(\mathbf{Q}, \mathbf{a}))) = \mathsf{tag}(\mathbf{Q}, \mathbf{a}, t) \oplus t) \\
&\qquad \wedge (\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{Q}, \mathbf{a}, t) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \mathbf{a}, t)))) \wedge (\mathbf{Q} = \mathbf{b})] \\
&= \sum_{\mathbf{b}} \Pr[(\mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{\tau_0}(x(\mathbf{b}, \mathbf{a}, t)) \oplus \mathsf{Hash}_{\tau_0}(x^{(\star)}(\mathbf{b}, \mathbf{a}))) = \mathsf{tag}(\mathbf{b}, \mathbf{a}, t) \oplus t) \\
&\qquad \wedge (\mathsf{msb}_{\lambda_0}(b) = \mathsf{tag}(\mathbf{b}, \mathbf{a}, t) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{\tau_0}(x(\mathbf{b}, \mathbf{a}, t)))) \wedge (\mathbf{Q} = \mathbf{b})]
\end{aligned}$$

To simplify notation, we write $x(\mathbf{b}, \mathbf{a}, t)$ as x, $x^{\star}(\mathbf{b}, \mathbf{a})$ as $x^{\star}$ and $\mathsf{tag}(\mathbf{b}, \mathbf{a}, t)$ as tag. So, we have

$$\begin{aligned}
&\sum_{\mathbf{b}} \Pr[(\mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{\tau_0}(x) \oplus \mathsf{Hash}_{\tau_0}(x^{(\star)})) = \mathsf{tag} \oplus t) \\
&\qquad \wedge (\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag} \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{\tau_0}(x))) \wedge (\mathbf{Q} = \mathbf{b})] \\
&= \sum_{\mathbf{b},a} \Pr[(\mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{a}(x) \oplus \mathsf{Hash}_{a}(x^{(\star)})) = \mathsf{tag} \oplus t) \\
&\qquad \wedge (\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag} \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{a}(x))) \wedge (\mathbf{Q} = \mathbf{b}) \wedge (\tau_0 = a)] \\
&= \sum_{\mathbf{b},a} \Pr[(\mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{a}(x) \oplus \mathsf{Hash}_{a}(x^{(\star)})) = \mathsf{tag} \oplus t) \wedge (\tau_0 = a)] \\
&\qquad \times \Pr[(\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag} \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{a}(x))) \wedge (\mathbf{Q} = \mathbf{b})] \\
&= \sum_{\mathbf{b},a} \Pr[(\mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{a}(x) \oplus \mathsf{Hash}_{a}(x^{(\star)})) = \mathsf{tag} \oplus t) \wedge (\tau_0 = a)] \\
&\qquad \times \left(\sum_{c_1} \Pr[(Q = (\mathsf{tag} \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{a}(x)))||c_1) \wedge (\mathbf{Q} = \mathbf{b})] \right)
\end{aligned}$$

Let $b = (\mathsf{tag} \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_a(x)))||c_1)$. Then $\Pr[(Q = b) \wedge (\mathbf{Q} = \mathbf{b})]$ is bounded from above by the q-interpolation probability of f. So, we have

$$\begin{aligned}
&\sum_{\mathbf{b},a} \Pr[(\mathsf{msb}_{\lambda_0}(\mathsf{Hash}_a(x) \oplus \mathsf{Hash}_a(x^{(\star)})) = \mathsf{tag} \oplus t) \wedge (\tau_0 = a)] \\
&\qquad\qquad \times \left(\sum_{c_1} \Pr[(Q = b) \wedge (\mathbf{Q} = \mathbf{b})] \right) \\
&\leq \sum_{\mathbf{b},a} \Pr[(\mathsf{msb}_{\lambda_0}(\mathsf{Hash}_a(x) \oplus \mathsf{Hash}_a(x^{(\star)})) = \mathsf{tag} \oplus t) \wedge (\tau_0 = a)] \times 2^{n-\lambda_0}\delta_q/(2^n)^q \\
&= 2^{n-\lambda_0}\delta_q/(2^n)^q \times \sum_{\mathbf{b},a} \Pr[(\mathsf{msb}_{\lambda_0}(\mathsf{Hash}_a(x) \oplus \mathsf{Hash}_a(x^{(\star)})) = \mathsf{tag} \oplus t) \wedge (\tau_0 = a)] \\
&= 2^{n-\lambda_0}\delta_q/(2^n)^q \times \sum_{\mathbf{b}} \Pr[\mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{\tau_0}(x) \oplus \mathsf{Hash}_{\tau_0}(x^{(\star)})) = \mathsf{tag} \oplus t] \\
&= 2^{n-\lambda_0}\delta_q/(2^n)^q \times \sum_{\mathbf{b}} \sum_{c_2} \Pr[\mathsf{Hash}_{\tau_0}(x) \oplus \mathsf{Hash}_{\tau_0}(x^{(\star)}) = (\mathsf{tag} \oplus t)||c_2] \\
&\leq 2^{n-\lambda_0}\delta_q/(2^n)^q \times \sum_{\mathbf{b}} 2^{n-\lambda_0}\varepsilon(\ell, \ell^{(\star)}) \\
&= 2^{n-\lambda_0}\delta_q/(2^n)^q \times (2^n)^{q-1} \times 2^{n-\lambda_0}\varepsilon(\ell, \ell^{(\star)}) \\
&= 2^{n-2\lambda_0}\delta_q\varepsilon(\ell, \ell^{(\star)}). \qquad (32)
\end{aligned}$$

Combining (31) and (32), we have,

$$\begin{aligned}
&\Pr[\mathsf{msb}_{\lambda_0}(Q \oplus \mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \boldsymbol{\tau}, \mathsf{tag}^{(\star)}))) = \mathsf{tag}(\mathbf{Q}, \boldsymbol{\tau}, \mathsf{tag}^{(\star)})] \\
&\leq \sum_{\mathbf{a}} \Big(\sum_{t} 2^{n-2\lambda_0}\delta_q\varepsilon(\ell, \ell^{(\star)}) \Big) \times \Pr[\boldsymbol{\tau} = \mathbf{a}] \\
&= \sum_{t} 2^{n-2\lambda_0}\delta_q\varepsilon(\ell, \ell^{(\star)}) \times \sum_{\mathbf{a}} \Pr[\boldsymbol{\tau} = \mathbf{a}] \\
&= 2^{\lambda_0}2^{n-2\lambda_0}\delta_q\varepsilon(\ell, \ell^{(\star)}) \\
&= 2^{n-\lambda_0}\varepsilon(\ell, \ell^{(\star)})\delta_q = \gamma/2^{\lambda_0}. \qquad (33)
\end{aligned}$$

This proves the second case.

□

Tightness of the security bound: The scheme NMAC is obtained as a variant of the Wegman-Carter scheme. The statement and proof of Theorem 1 follows the bound on the Wegman-Carter scheme established by Bernstein [5]. As mentioned earlier, Bernstein's bound has been proved to be tight [15, 18]. A natural question is to consider whether the bound of Theorem 1 is also tight. We have considered this question for NMAC. It does not seem possible to use the proof approach used in [15, 18] to show the tightness of the bound in Theorem 1. In fact, the approach does not also seem to work for the generic scheme NMAC-Generic.

4.1 Reducing Key Size

In a practical instantiation of NMAC, the random function f will be instantiated by a keyed function F_K. The key for the entire scheme will consist of the key K along with the $\#\mathcal{L}$ keys $(\tau_\lambda)_{\lambda \in \mathcal{L}}$ for

the hash function Hash. Depending on the size of $\mathcal{L}$, for certain applications, the size of the key may be too large. Our next constructions show how to obtain NMAC schemes with short keys.

The hash family $\{\mathsf{Hash}_\tau\}_{\tau\in\Theta}$, the nonce space $\mathcal{N}$, the message space $\mathcal{M}$, the set of allowed tag lengths $\mathcal{L}$ and the tag space remain the same as in the case of NMAC.

Our goal is to derive the key for the hash function by applying a PRF to the concatenation of the tag length and the nonce. Depending upon the actual choice of the hash function, the key could either be an n-bit string (or, a string of some fixed length which is at least n), or, it could be a variable length string which depends upon the length of the message. Typical examples of hash function where the key is a fixed length string is the polynomial hash or the BRW hash [4, 17, 6] while typical examples of hash function where the key depends upon the length of the message is either the multi-linear hash [13], or the pseudo-dot product [27], or the UMAC [8] construction.

We consider the key of the hash function to be a sequence of n-bit blocks with the last block possibly being a partial block. Given the hash function Hash and a message x, let $\mathfrak{b}(x)$ denote the number of n-bit blocks of key material required by Hash to process the message x. As mentioned above, depending upon the choice of Hash, $\mathfrak{b}(x)$ could be independent of x (i.e., Hash uses fixed length keys), or, it could depend upon x (i.e., Hash uses a key which depends upon the length of x).

We start by constructing a nonce-based MAC scheme from a stream cipher supporting an initialisation vector. The assumption on such a stream cipher is that it is a PRF [2]. Formally, we use the PRF $\{\mathsf{SC}_K\}_{K\in\mathcal{K}}$, where SC_K is a stream cipher which maps an n-bit string under the key K to an output keystream. We will assume that the output keystream is of some fixed length which is sufficiently big for all practical applications. An appropriate length prefix of the output keystream is used in a particular context. We denote the NMAC scheme built from SC as SC-NMAC. The tag generation algorithm for the SC-NMAC scheme is shown in Table 3. The verification algorithm $\mathsf{SC\text{-}NMAC.Verify}_K(N, x, \mathsf{tag}, \lambda)$ works as follows: compute $\mathsf{tag}' = \mathsf{SC\text{-}NMAC.Gen}_K(N, x, \lambda)$; return true if $\mathsf{tag} = \mathsf{tag}'$, else return false.

The key space for SC-NMAC is $\mathcal{K}$. The key generation algorithm consists of sampling K uniformly at random from $\mathcal{K}$.

Table 3: A secure and efficient NMAC scheme using a stream cipher supporting an initialisation vector.

$\mathsf{SC\text{-}NMAC.Gen}_K(N, x, \lambda)$
$b = \mathfrak{b}(x)$;
$(Q, \tau) = \mathsf{msb}_{(b+1)n}(\mathsf{SC}_K(\mathsf{bin}_8(\lambda)||N))$;
$R = Q \oplus \mathsf{Hash}_\tau(x)$;
$\mathsf{tag} = \mathsf{msb}_\lambda(R)$;
return tag.

The security of SC-NMAC is given by the following result.

Theorem 2. *In* SC-NMAC *defined in Table 3, suppose that the hash function* $\{\mathsf{Hash}_\tau\}_{\tau\in\Theta}$ *is* ε*-AXU, where* $\varepsilon(\ell, \ell') \geq 1/2^n$ *for all* $\ell, \ell' \leq L$.

Fix a query profile $\mathfrak{C}$. *For* $\lambda \in \mathcal{L}$, *let* $q_{g,\lambda}$ *(resp.* $q_{v,\lambda}$*) be the number of tag generation (resp. verification) queries for* λ *which are in* $\mathfrak{C}$. *Let* $q_g = \sum_{\lambda\in\mathcal{L}} q_{g,\lambda}$ *and* $q_v = \sum_{\lambda\in\mathcal{L}} q_{v,\lambda}$. *Let* σ_g *(resp.* σ_v*) be the total number of bits in all the tag generation (resp. verification) queries in* $\mathfrak{C}$.

Let λ be such that $q_{v,\lambda} \geq 1$ and for $1 \leq i \leq q_{v,\lambda}$, let $Q_{v,\lambda}^{(i)} = (N_{v,\lambda}^{(i)}, x_{v,\lambda}^{(i)}, \mathsf{tag}_{v,\lambda}^{(i)}, \lambda)$ be the i-th verification query with tag length λ. Let $\ell_{v,\lambda}^{(i)} = \mathsf{len}(x_{v,\lambda}^{(i)})$. Corresponding to $Q_{v,\lambda}^{(i)}$, there is at most one tag generation query $Q_{g,\lambda}^{(i^\star)} = (N_{g,\lambda}^{(i^\star)}, x_{g,\lambda}^{(i^\star)}, \lambda)$ such that $N_{v,\lambda}^{(i)} = N_{g,\lambda}^{(i^\star)}$. Let $\ell_{g,\lambda}^{(i^\star)} = \mathsf{len}(x_{g,\lambda}^{(i^\star)})$ if there is such a $Q_{g,\lambda}^{(i^\star)}$, otherwise $\ell_{g,\lambda}^{(i^\star)}$ is undefined.

Fix $\lambda_0 \in \mathcal{L}$. Let $\mathcal{S}_{\lambda_0}$ be the set of all queries made by the adversary other than the verification queries for tag length λ_0. Suppose that the queries in $\mathcal{S}_{\lambda_0}$ give rise to at most q distinct (nonce, tag-length) values. Then

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{SC\text{-}NMAC}}[\lambda_0](t, \mathfrak{C}) \;\leq\; \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(t + t', q_g + q_v, n(q_g + q_v)) + \frac{1}{2^{\lambda_0}} \times \sum_{1 \leq i \leq q_{v,\lambda_0}} \gamma_i \qquad (34)$$

where $\gamma_i = 2^n \varepsilon(\ell_{v,\lambda_0}^{(i)}, \ell_{g,\lambda_0}^{(i^\star)})$ if there is a $Q_{g,\lambda_0}^{(i^\star)}$ corresponding to $Q_{v,\lambda_0}^{(i)}$ with $N_{v,\lambda_0}^{(i)} = N_{g,\lambda_0}^{(i^\star)}$; otherwise $\gamma_i = 1$. Here t' is the time required to hash $q_v + q_g$ messages of total length at most $\sigma_g + \sigma_v$, plus some bookkeeping time.

Proof. The proof is similar to the proof of Theorem 1. We mention the differences.

The first reduction is to replace SC_K by a uniform random function ρ from $\{0,1\}^n$ to $\{0,1\}^L$. The advantage of the adversary in detecting this change is captured by the term $\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(t + t', q_g + q_v, n(q_g + q_v))$ in (34). Let the scheme resulting from the replacement be denoted as ρ-NMAC.

Since SC_K has been taken care of, the ensuing analysis is information theoretic. Let $\mathcal{A}$ be a deterministic and computationally unbounded adversary attacking ρ-NMAC and having query profile $\mathfrak{C}$. It is required to upper bound $\mathsf{Adv}^{\mathrm{auth}}_{\rho\text{-}\mathsf{NMAC}}[\lambda_0](\mathcal{A})$.

As in the proof of Theorem 1, the task reduces to analysing the probability of the event $\mathsf{succ}(\mathcal{A}(\mathcal{T}), \lambda_0)$ for a transcript $\mathcal{T}$ whose query profile is $\mathfrak{C}$.

The second reduction is to assume that $q_{v,\lambda_0} = 1$; *the third reduction* is to assume that all queries after the single verification query for tag length λ_0 are discarded. These reductions are also used in the proof of Theorem 1 and the justifications for these reductions in the present context are the same as those described in the proof of Theorem 1. As in Theorem 1, consider the set $\mathcal{S}_{\lambda_0}$ which consists of all queries made by $\mathcal{A}$ other than the verification queries for λ_0. Further, similar to the proof of Theorem 1, insert queries to the transcript $\mathcal{T}$, to ensure that the number of distinct (nonce, tag-length) pairs arising from the queries in $\mathcal{S}_{\lambda_0}$ is q.

In view of the above reductions, it is sufficient to consider an adversary $\mathcal{A}$ with a transcript $\mathcal{T}$ where the last query is the verification query $(N, x, \mathsf{tag}, \lambda_0)$ for tag length λ_0. Also, let $(N^{(1)}, \lambda^{(1)}), \ldots, (N^{(q)}, \lambda^{(q)})$ be the distinct (nonce, tag-length) pairs arising from the queries in $\mathcal{S}_{\lambda_0}$. For $1 \leq i \leq q$, define $\mathfrak{N}^{(i)} = \mathsf{bin}_8(\lambda^{(i)}) || N^{(i)}$, $(Q^{(i)}, \tau_i) = \rho(\mathfrak{N}^{(i)})$ (considering the full length output of ρ), $\mathbf{Q} = (Q^{(1)}, \ldots, Q^{(q)})$ and $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_q)$. The entire randomness in the transcript arises from $\mathbf{Q}$ and $\boldsymbol{\tau}$.

At this point, we would like to mention a small difference with the proof of Theorem 1. In the scheme NMAC, the hash key depends upon the tag length, whereas in SC-NMAC, the hash key is determined by (nonce, tag-length) pair. As a consequence, the vector $\boldsymbol{\tau}$ defined above has q components, while the vector $\boldsymbol{\tau}$ defined in the proof of Theorem 1 has q' components, where q' is the number of distinct tag lengths arising from the queries in $\mathcal{S}_{\lambda_0}$.

Modulo this small difference, the rest of the proof is the same as the proof of Theorem 1. In particular, the proof divides into two cases. The first case is where the adversary does not make any previous tag generation query with (nonce, tag-length) pair equal to (N, λ_0) and the second case is where the adversary does make such a query. The probability calculations for these two

cases are almost the same as those in the proof of Theorem 1. The only difference is that in the present case, ρ is uniform random function and so $\delta_j = 1$. Using these values of δ_j, the calculations done in the two cases of the proof of Theorem 1 show the bound stated in (34).

□

In the scheme SC-NMAC, the pair (Q, τ) is derived by applying the stream cipher to $\mathsf{bin}_8(\lambda)||N$. Since a stream cipher produces a long enough keystream, a single application of SC is sufficient to obtain the pair (Q, τ). Suppose that we wish to use a PRF F whose output is an n-bit string (or, a short fixed length string). Clearly, then a single invocation of F will not be sufficient to obtain the pair (Q, τ). The PRF F will have to be invoked repeatedly to obtain an output bit string of desired length from which the pair (Q, τ) can be obtained.

Formally, we use a PRF family $\{\mathsf{F_K}\}_{K \in \mathcal{K}}$, where for each $K \in \mathcal{K}$, $\mathsf{F}_K : \{0,1\}^n \rightarrow \{0,1\}^n$. Similar to the case of SC-NMAC, the hash family $\{\mathsf{Hash}_\tau\}_{\tau \in \Theta}$, the nonce space $\mathcal{N}$, the message space $\mathcal{M}$, the set of allowed tag lengths $\mathcal{L}$ and the tag space remain the same as in the case of NMAC. The key space for the scheme is $\mathcal{K}$. The key generation algorithm consists of sampling K uniformly at random from $\mathcal{K}$.

The tag generation algorithm of an NMAC scheme built from the PRF F is shown in Table 4 and is denoted as F-NMAC.Gen. The verification algorithm $\mathsf{F\text{-}NMAC.Verify}(N, x, \mathsf{tag}, \lambda)$ works as follows. Given $(N, x, \mathsf{tag}, \lambda)$, compute $\mathsf{tag}' = \mathsf{F\text{-}NMAC.Gen}_K(N, x, \lambda)$; if $\mathsf{tag} = \mathsf{tag}'$, return true, else return false. In Table 4, F is used in a counter type mode of operation which was proposed in [25].

Instantiation of F may be done by a fixed output length PRF such as Siphash [1]. Alternatively, it can also be done using the encryption function $E_K(\cdot)$ of a block cipher. Since E is a bijection, the PRF assumption on $E_K(\cdot)$ does not hold beyond the birthday bound. While using $E_K(\cdot)$, it would have been better to perform the analysis under the assumption that $E_K(\cdot)$ is a pseudo-random permutation (PRP). This, however, is problematic. The key τ to the hash function is derived by applying $E_K(\cdot)$. Under the assumption that $E_K(\cdot)$ is a PRP, it would not be possible to assume that τ is uniformly distributed. The differential probability determining the AXU property of the hash function is computed based on uniform random τ. So, if τ cannot be considered to be uniform random, the AXU property of the hash function cannot be invoked. As a result, the proof would not go through. On the other hand, up to the birthday bound, it is reasonable to assume that the encryption function of a secure block cipher behaves like a PRF.

Table 4: A secure and efficient NMAC scheme using a short output length PRF.

$\mathsf{F\text{-}NMAC.Gen}_K(N, x, \lambda)$
$b = \mathfrak{b}(x)$;
$\mathsf{S} = \mathsf{F}_K(\mathsf{bin}_8(\lambda)||N)$;
$(Q, \tau) = \mathsf{F}_K(\mathsf{S} \oplus \mathsf{bin}_n(1))|| \cdots ||\mathsf{F}_K(\mathsf{S} \oplus \mathsf{bin}_n(b+1))$;
$R = Q \oplus \mathsf{Hash}_\tau(x)$;
$\mathsf{tag} = \mathsf{msb}_\lambda(R)$;
return tag.

The security of F-NMAC is given by the following result.

Theorem 3. *In* F-NMAC *defined in Table 4, suppose that the hash function* $\{\mathsf{Hash}_\tau\}_{\tau \in \Theta}$ *is* ε*-AXU, where* $\varepsilon(\ell, \ell') \geq 1/2^n$ *for all* $\ell, \ell' \leq L$.

Fix a query profile $\mathfrak{C}$. For $\lambda \in \mathcal{L}$, let $q_{g,\lambda}$ (resp. $q_{v,\lambda}$) be the number of tag generation (resp. verification) queries for λ which are in $\mathfrak{C}$. Let $q_g = \sum_{\lambda\in\mathcal{L}} q_{g,\lambda}$ and $q_v = \sum_{\lambda\in\mathcal{L}} q_{v,\lambda}$. Let σ_g (resp. σ_v) be the total number of bits in all the tag generation (resp. verification) queries in $\mathfrak{C}$.

Let λ be such that $q_{v,\lambda} \geq 1$ and for $1 \leq i \leq q_{v,\lambda}$, let $Q_{v,\lambda}^{(i)} = (N_{v,\lambda}^{(i)}, x_{v,\lambda}^{(i)}, \mathsf{tag}_{v,\lambda}^{(i)}, \lambda)$ be the i-th verification query with tag length λ. Let $\ell_{v,\lambda}^{(i)} = \mathsf{len}(x_{v,\lambda}^{(i)})$. Corresponding to $Q_{v,\lambda}^{(i)}$, there is at most one tag generation query $Q_{g,\lambda}^{(i^\star)} = (N_{g,\lambda}^{(i^\star)}, x_{g,\lambda}^{(i^\star)}, \lambda)$ such that $N_{v,\lambda}^{(i)} = N_{g,\lambda}^{(i^\star)}$. Let $\ell_{g,\lambda}^{(i^\star)} = \mathsf{len}(x_{g,\lambda}^{(i^\star)})$ if there is such a $Q_{g,\lambda}^{(i^\star)}$, otherwise $\ell_{g,\lambda}^{(i^\star)}$ is undefined.

Fix $\lambda_0 \in \mathcal{L}$. Let $\mathcal{S}_{\lambda_0}$ be the set of all queries made by the adversary other than the verification queries for tag length λ_0. Suppose that the queries in $\mathcal{S}_{\lambda_0}$ give rise to at most q distinct (nonce, tag-length) values. Then

$$\begin{aligned}\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{F\text{-}NMAC}}[\lambda_0](t,\mathfrak{C}) \;\leq\; & \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(t+t', B_g+B_v, n(B_g+B_v)) \\ & + \frac{(B_g+B_v)^2}{2^n} + \frac{1}{2^{\lambda_0}} \times \sum_{1\leq i\leq q_{v,\lambda_0}} \gamma_i \end{aligned} \tag{35}$$

where

- *$\gamma_i = 2^n \varepsilon(\ell_{v,\lambda_0}^{(i)}, \ell_{g,\lambda_0}^{(i^\star)})$ if there is a $Q_{g,\lambda_0}^{(i^\star)}$ corresponding to $Q_{v,\lambda_0}^{(i)}$ with $N_{v,\lambda_0}^{(i)} = N_{g,\lambda_0}^{(i^\star)}$; otherwise $\gamma_i = 1$;*
- *$b_{v,\lambda}^{(i)} = \mathfrak{b}(x_{v,\lambda}^{(i)})$, $b_{g,\lambda}^{(i)} = \mathfrak{b}(x_{g,\lambda}^{(i)})$, $B_v = \sum_{\lambda} \sum_{1\leq i\leq q_{v,\lambda}} (b_{v,\lambda}^{(i)} + 2)$ and $B_g = \sum_{\lambda} \sum_{1\leq i\leq q_{g,\lambda}} (b_{g,\lambda}^{(i)} + 2)$.*

Here t' is the time required to hash $q_v + q_g$ messages of total length at most $\sigma_g + \sigma_v$, plus some bookkeeping time.

Proof. The proof is very similar to the proofs of Theorems 1 and 2. We briefly discuss the differences. There are two differences in the bound.

The first difference is in the number of queries to the PRF F in the expression $\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}$. In the present case, if a query requires $b+1$ n-bit blocks to obtain the pair (Q,τ), the number of times F is invoked is $b+2$. The rest of the analysis proceeds by replacing F with a uniform random function ρ from $\{0,1\}^n$ to $\{0,1\}^n$.

The main argument requires that for distinct values of (N,λ), the random variables (Q,τ) are independent and uniformly distributed. The pair (Q,τ) is derived by successively applying ρ to $\mathsf{S} \oplus \mathsf{bin}_n(1), \ldots, \mathsf{S} \oplus \mathsf{bin}_n(b+1)$ where S itself is obtained by applying ρ to $\mathsf{bin}_8(\lambda)||N$. If for distinct values of (N,λ), the quantities $\mathsf{S}, \mathsf{S} \oplus \mathsf{bin}_n(1), \ldots, \mathsf{S} \oplus \mathsf{bin}_n(b+1)$ are distinct, then the independent and uniform random distribution of (Q,τ) is ensured.

Let the q distinct values of (nonce, tag-length) pairs arising from the queries in $\mathcal{S}_{\lambda_0}$ be $(N^{(1)},\lambda^{(1)})$, $\ldots$, $(N^{(q)},\lambda^{(q)})$. Let $\mathcal{D}^{(i)} = \{\mathsf{S}^{(i)}, \mathsf{S}^{(i)} \oplus \mathsf{bin}_n(1), \ldots, \mathsf{S}^{(i)} \oplus \mathsf{bin}_n(b^{(i)}+1)\}$ be the set of random variables in the input of ρ corresponding to $(N^{(i)},\lambda^{(i)})$. Let $\mathcal{D} = \cup_{i=1}^{q} \mathcal{D}^{(i)}$ and so $\#\mathcal{D} \leq B_g + B_v$. Let bad be the event that any two of the variables in $\mathcal{D}$ are equal. Using the fact that ρ is a uniform random function, it is standard to see that $\Pr[\mathsf{bad}] \leq (B_g+B_v)^2/2^n$.

Let $\mathcal{A}$ be an adversary attacking the scheme where F is replaced with ρ. We assume that $\mathcal{A}$ is deterministic and computationally unbounded. Let $\mathsf{succ}(\mathcal{A})$ be the event that an adversary $\mathcal{A}$ is successful. Then

$$\begin{aligned}\Pr[\mathsf{succ}(\mathcal{A})] \;&\leq\; \Pr[\mathsf{bad}] + \Pr[\mathsf{succ}(\mathcal{A})|\overline{\mathsf{bad}}] \\ &\leq\; \frac{(B_g+B_v)^2}{2^n} + \Pr[\mathsf{succ}(\mathcal{A})|\overline{\mathsf{bad}}].\end{aligned}$$

Table 5: A variable tag length MAC scheme obtained from a PRF $\{\mathsf{G}_K\}_{K\in\mathcal{K}}$, $G_K : \mathcal{M}\times\mathcal{L} \to \{0,1\}^n$.

$\mathsf{MAC.Gen}_K(x,\lambda)$
  $R = \mathsf{G}_K(x,\lambda)$;
  $\mathsf{tag} = \mathsf{msb}_\lambda(R)$;
return tag.

Conditioned on the event $\overline{\mathsf{bad}}$, the pairs $(Q^{(i)}, \tau^{(i)})$ are independent and uniformly distributed. From this point onwards, the rest of the proof is exactly the same as the proof of Theorem 2 and provides the same bound. We skip these details. □

5 PRF Supporting Variable Input Length

There could be situations where it is not feasible to use a nonce. Such situations require the use of MAC schemes which do not utilise nonces. The goal of supporting variable length tag, however, remains the same. This calls for MAC schemes which do not use nonces but support variable length tags.

Suppose as before that the message space is $\mathcal{M}$ and the set of allowed tag lengths is $\mathcal{L}$. Let $\{\mathsf{G}_K\}_{K\in\mathcal{K}}$, $\mathsf{G}_K : \mathcal{M}\times\mathcal{L} \to \{0,1\}^n$ be a PRF. Such a PRF can be used to obtain a MAC scheme supporting variable length tag. The tag generation algorithm for the scheme MAC is shown in Table 5. The verification algorithm $\mathsf{MAC.Verify}_K(x,\mathsf{tag},\lambda)$ works as follows: compute $\mathsf{tag}' = \mathsf{MAC.Gen}_K(x,\lambda)$; if $\mathsf{tag} = \mathsf{tag}'$, return true, else return false. The key space for MAC is $\mathcal{K}$ and the key generation algorithm consists of choosing K uniformly at random from $\mathcal{K}$.

The following result relates the authenticity of MAC to the PRF-security of G.

Theorem 4. *Fix a query profile $\mathfrak{C}$. For $\lambda \in \mathcal{L}$, let $q_{g,\lambda}$ (resp. $q_{v,\lambda}$) be the number of tag generation (resp. verification) queries for λ which are in $\mathfrak{C}$. Let $q_g = \sum_{\lambda\in\mathcal{L}} q_{g,\lambda}$ and $q_v = \sum_{\lambda\in\mathcal{L}} q_{v,\lambda}$. Let σ_g (resp. σ_v) be the total number of bits in all the tag generation (resp. verification) queries in $\mathfrak{C}$. Then for the scheme MAC given in Table 5,*

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{MAC}}[\lambda](t,\mathfrak{C}) \;\leq\; \frac{q_{v,\lambda}}{2^\lambda} + \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{G}}(t+t', q, \sigma_g+\sigma_v) \tag{36}$$

where t' denotes bookkeeping time.

Proof. Let $\mathcal{A}$ be an adversary attacking the authentication property of MAC. Using $\mathcal{A}$, we build an adversary $\mathcal{B}$ attacking the PRF-property of G as follows. $\mathcal{B}$ has access to an oracle, which is either the real oracle (i.e., $\mathsf{G}_K(\cdot,\cdot)$ where K is chosen uniformly from $\mathcal{K}$) or the random oracle, which on distinct inputs (x,λ) returns independent and uniform random n-bit strings. Let the response from the oracle of $\mathcal{B}$ corresponding to (x,λ) be str.

$\mathcal{B}$ maintains a list $\mathfrak{L}$ of tuples of the form (x,λ,str) which is initially empty. For any tag generation query (x_g,λ_g) made by $\mathcal{A}$, $\mathcal{B}$ queries its oracle on the same input and receives in return an n-bit string str_g. It enters $(x_g,\lambda_g,\mathsf{str}_g)$ in $\mathfrak{L}$ and returns $\mathsf{msb}_{\lambda_g}(\mathsf{str}_g)$ to $\mathcal{A}$. For any verification query $(x_v,\mathsf{tag}_v,\lambda_v)$ made by $\mathcal{A}$, $\mathcal{B}$ first searches for a tuple containing (x_v,λ_v) in $\mathfrak{L}$. If no such tuple is present in $\mathfrak{L}$, $\mathcal{B}$ makes a query to its oracle on (x_v,λ_v). Let the corresponding n-bit string it obtains in return, either from $\mathfrak{L}$ or from its oracle, be str_v. If str_v is obtained from the oracle, i.e. if

the tuple $(x_v, \lambda_v, \mathsf{str}_v)$ is not present in $\mathfrak{L}$, $\mathcal{B}$ adds it to $\mathfrak{L}$ and returns true to $\mathcal{A}$ if $\mathsf{msb}_{\lambda_v}(\mathsf{str}_v) = \mathsf{tag}_v$; otherwise, $\mathcal{B}$ returns false to $\mathcal{A}$. If for any of the verification queries $(x_v, \mathsf{tag}_v, \lambda_v)$ it turns out that $\lambda_v = \lambda$ and $\mathcal{B}$ returns true to $\mathcal{A}$ then $\mathcal{B}$ finally outputs 1; otherwise $\mathcal{B}$ continues and when $\mathcal{A}$ terminates, $\mathcal{B}$ returns 0. Note that by the condition on $\mathcal{A}$ all its tag generation queries are distinct and further (x_v, λ_v) corresponding to any verification query is also distinct from the tag generation queries. But for two verification queries the part (x_v, λ_v) may repeat. By using $\mathfrak{L}$, $\mathcal{B}$ ensures that all its queries to its oracles are distinct for any combination of queries made by $\mathcal{A}$. Let q'_v be the number of verification queries made by $\mathcal{A}$ with unique pair of message and tag length and σ'_v be the total number of bits in these q'_v queries. Clearly, $q_g + q'_v \leq q_g + q_v = q$ and $\sigma_g + \sigma'_v \leq \sigma_g + \sigma_v$. Hence, $\mathcal{B}$ makes at most q queries to its oracle with total number of bits at most $\sigma_g + \sigma_v$.

Let $\mathcal{B}^{\mathsf{real}} \Rightarrow 1$ ($\mathcal{B}^{\mathsf{rnd}} \Rightarrow 1$) be the event that $\mathcal{B}$ returns 1 when its oracle is real (resp. random). By definition,

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{G}}(\mathcal{B}) \;=\; \Pr[\mathcal{B}^{\mathsf{real}} \Rightarrow 1] - \Pr[\mathcal{B}^{\mathsf{rnd}} \Rightarrow 1]. \tag{37}$$

Also, $\mathcal{B}$ returns 1 if and only if some verification query made by $\mathcal{A}$ for tag length λ returns true. So,

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{MAC}}[\lambda](\mathcal{A}) \;=\; \Pr[\mathcal{B}^{\mathsf{real}} \Rightarrow 1]. \tag{38}$$

When the oracle to $\mathcal{B}$ is random, the outputs that $\mathcal{B}$ receives from the oracle are independent and uniform random n-bit strings. In this case, $\mathcal{B}$ outputs 1 if and only if at least one of the $q_{v,\lambda}$ verification queries for tag length λ is successful. The probability that any such query is successful is $1/2^\lambda$ and so the probability that $\mathcal{B}$ outputs 1 when its oracle is random is at most $q_{v,\lambda}/2^\lambda$. Combining this fact with (37) and (38) we obtain

$$\begin{aligned} \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{G}}(\mathcal{B}) &= \Pr[\mathcal{B}^{\mathsf{real}} \Rightarrow 1] - \Pr[\mathcal{B}^{\mathsf{rnd}} \Rightarrow 1] \\ &\geq \mathsf{Adv}^{\mathrm{auth}}_{\mathsf{MAC}}[\lambda](\mathcal{A}) - \frac{q_{v,\lambda}}{2^\lambda}. \end{aligned} \tag{39}$$

From previous discussion on the number of queries and the query complexities of $\mathcal{A}$ and $\mathcal{B}$, we obtain the following relation.

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{MAC}}[\lambda](t, q_g, q_v, \sigma_g, \sigma_v) \;\leq\; \frac{q_{v,\lambda}}{2^\lambda} + \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{G}}(t + t', q_g + q_v, \sigma_g + \sigma_v). \tag{40}$$

□

In view of Theorem 4, the problem reduces to constructing a PRF which accepts as input a pair (message, tag-length) and produces as output an n-bit string. Such a PRF can be constructed using known techniques. We consider the construction of such a PRF using a hash function and a PRF which accepts n-bit strings as inputs. Table 6 provides three such constructions. The message space, the set of allowed tag lengths and the tag space are the same as those of NMAC. Further, fStr is a fixed binary string of length $n - 8$ bits.

All the three PRF constructions in Table 6 utilise a hash function $\{\mathsf{Hash}_\tau\}_{\tau \in \Theta}$. The scheme F-PRF uses a PRF $\{\mathsf{F}_K\}_{K \in \mathcal{K}}$, $\mathsf{F}_K : \{0,1\}^n \to \{0,1\}^n$. The key to F-PRF consists of the pair (K, τ) consisting of the key for F and the key for Hash. The schemes SC-PRF and E-PRF use a single key and derive the key for the hash function as part of the scheme. The scheme SC-PRF uses a stream cipher supporting an initialisation vector which is modelled as a PRF $\{\mathsf{SC}_K\}_{K \in \mathcal{K}}$,

Table 6: PRF schemes from a hash function $\{\mathsf{Hash}_\tau\}_{\tau\in\Theta}$ and a PRF which accepts n-bit strings as inputs.

$\mathsf{F\text{-}PRF}_{K,\tau}(x,\lambda)$
$N = \mathsf{Hash}_\tau(\mathsf{bin}_8(\lambda)\|\|x);$
$R = \mathsf{F}_K(N);$
return R.
	$\mathsf{SC\text{-}PRF}_K(x,\lambda)$
	$\mathsf{str} = \mathsf{bin}_8(\lambda)\|\|\mathsf{fStr};$
	$b = \mathfrak{b}(x);$
	$\tau = \mathsf{msb}_{bn}(\mathsf{SC}_K(\mathsf{str}));$
	$N = \mathsf{Hash}_\tau(x);$
	$R = \mathsf{SC}_K(N);$
	return R
		$\mathsf{E\text{-}PRF}_K(x,\lambda)$
		$\mathsf{str} = \mathsf{bin}_8(\lambda)\|\|\mathsf{fStr};$
		$b = \mathfrak{b}(x);$
		if $b = 1$
		$\tau = \mathsf{E}_K(\mathsf{str});$
		else
		$\mathsf{Q} = \mathsf{E}_K(\mathsf{str});$
		for $i \leftarrow 1$ to b do
		$\mathsf{Q}_i = \mathsf{E}_K(\mathsf{Q} \oplus \mathsf{bin}_n(i));$
		$\tau = \mathsf{Q}_1\|\|\cdots\|\|\mathsf{Q}_b;$
		$N = \mathsf{Hash}_\tau(x);$
		$R = \mathsf{E}_K(N);$
		return R.

$\mathsf{SC}_K : \{0,1\}^n \to \{0,1\}^L$ where L is large enough to derive all practical size strings. The scheme E-PRF uses a PRF $\{\mathsf{E}_K\}_{K\in\mathcal{K}}$, $\mathsf{E}_K : \{0,1\}^n \to \{0,1\}^n$.

The key space for F-PRF is $\mathcal{K} \times \Theta$ and choosing a key (K,τ) consists of sampling K uniformly at random from $\mathcal{K}$ and independently sampling τ uniformly at random from Θ. The key spaces for SC-PRF and E-PRF are both $\mathcal{K}$ and choosing a key consists of sampling K uniformly at random from $\mathcal{K}$.

The security statements and the proofs for the PRF schemes in Table 6 are given in Appendix B.

6 Conclusion

In this paper, we have considered the problem of constructing variable tag length MAC schemes. Several variants obtained from the Wegman-Carter MAC scheme have been shown to be insecure. One of these variants is proved to be secure. This scheme is extended to obtain constructions of single-key nonce-based variable tag length MAC schemes using either a stream cipher or a short-output PRF. The problem of constructing variable tag length MAC schemes without nonces have also been considered and several practical schemes have been described.

Acknowledgement

We thank Debrup Chakraborty for having provided an outline of the attack on NMAC-t1 and also for several other important discussions.

References

[1] Jean-Philippe Aumasson and Daniel J. Bernstein. Siphash: A fast short-input PRF. In Steven D. Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012.*

Proceedings, volume 7668 of *Lecture Notes in Computer Science*, pages 489–508. Springer, 2012.

[2] Côme Berbain and Henri Gilbert. On the security of IV dependent stream ciphers. In Alex Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 254–273. Springer, 2007.

[3] Daniel J. Bernstein. The Salsa20 family of stream ciphers. `http://cr.yp.to/papers.html#salsafamily`. Document ID: 31364286077dcdff8e4509f9ff3139ad. Date: 2007.12.25.

[4] Daniel J. Bernstein. The poly1305-aes message-authentication code. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*, pages 32–49. Springer, 2005.

[5] Daniel J. Bernstein. Stronger security bounds for Wegman-Carter-Shoup authenticators. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 164–180. Springer, 2005.

[6] Daniel J. Bernstein. Polynomial evaluation and message authentication, 2007. `http://cr.yp.to/papers.html#pema`.

[7] Daniel J. Bernstein and Tung Chou. Faster binary-field multiplication and faster binary-field macs. In Antoine Joux and Amr M. Youssef, editors, *Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers*, volume 8781 of *Lecture Notes in Computer Science*, pages 92–111. Springer, 2014.

[8] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. UMAC: Fast and secure message authentication. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 216–233. Springer, 1999.

[9] CAESAR. Competition for Authenticated Encryption: Security, Applicability, and Robustness. `http://competitions.cr.yp.to/caesar.html`.

[10] Debrup Chakraborty, Sebati Ghosh, and Palash Sarkar. A fast single-key two-level universal hash function. *IACR Trans. Symmetric Cryptol.*, 2017(1):106–128, 2017.

[11] Hal Finney. CFRG discussion on UMAC. `https://marc.info/?l=cfrg&m=143336318427069&w=2`, accessed on 15 November, 2019, 2005.

[12] Hal Finney. CFRG discussion on UMAC. `https://marc.info/?l=cfrg&m=143336318527072&w=2`, accessed on 15 November, 2019, 2005.

[13] Edgar N. Gilbert, F. Jessie MacWilliams, and Neil J. A. Sloane. Codes which detect deception. *Bell System Technical Journal*, 53:405–424, 1974.

[14] Ted Krovetz. UMAC: Message authentication code using universal hashing. `https://tools.ietf.org/html/draft-krovetz-umac-05.html`, accessed on 15 November, 2019., 2005.

[15] Atul Luykx and Bart Preneel. Optimal forgeries against polynomial-based macs and GCM. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 445–467. Springer, 2018.

[16] James H. Manger. Attacker changing tag length in OCB. `https://mailarchive.ietf.org/arch/msg/cfrg/gJtV9FCw92MguqqhxrSNUyIDZIw`, accessed on 15 November, 2019., 2013.

[17] David A. McGrew and John Viega. The security and performance of the Galois/Counter Mode (GCM) of operation. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004.

[18] Mridul Nandi. Bernstein bound on WCS is tight - repairing luykx-preneel optimal forgeries. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 213–238. Springer, 2018.

[19] Mike Ounsworth. Footguns as an axis of security analysis. `https://groups.google.com/a/list.nist.gov/forum/#!topic/pqc-forum/l2iYk-8sGnI`, accessed on 15 November, 2019, 2019.

[20] Reza Reyhanitabar, Serge Vaudenay, and Damian Vizár. Authenticated encryption with variable stretch. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 396–425, 2016.

[21] P. Rogaway and D. Wagner. A critique of ccm. Cryptology ePrint Archive, Report 2003/070, 2003. `https://eprint.iacr.org/2003/070`.

[22] Victor Shoup. On fast and provably secure message authentication based on universal hashing. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 313–328. Springer, 1996.

[23] UMAC. CFRG discussion on UMAC. `http://marc.info/?l=cfrg&m=143336318427068&w=2`, accessed on 15 November, 2019., 2005.

[24] David Wagner. CFRG discussion on UMAC. `https://marc.info/?l=cfrg&m=143336318527073&w=2`, accessed on 15 November, 2019, 2005.

[25] Peng Wang, Dengguo Feng, and Wenling Wu. HCTR: A variable-input-length enciphering mode. In Dengguo Feng, Dongdai Lin, and Moti Yung, editors, *CISC*, volume 3822 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2005.

[26] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981.

[27] Shmuel Winograd. A new algorithm for inner product. *IEEE Transactions on Computers*, 17:693–694, 1968.

A Attack on NMAC-t6

Algorithm 2 Attack on NMAC-t6 for $\lambda_2 = n$:

1: set $\lambda_2 \leftarrow n$;
2: choose $\lambda_1 \in \mathcal{L}$, such that $\lambda_1 < \lambda_2$;
3: choose any $N_1 \in \mathcal{N}$ and any $x_1 \in \mathcal{M}$ and let
4: $\quad \mathsf{tag}^{(1)} \leftarrow \mathcal{O}_g(N_1, x_1, \lambda_1)$;
5: set $\mathcal{D} \leftarrow \{\}$;
6: choose $x_2 \in \mathcal{M}$, such that $x_2 \neq x_1$;
7: **do**
8: $\quad$ choose $\mathsf{tag}^{(2)} \leftarrow \{0,1\}^{\lambda_1} \setminus \mathcal{D}$;
9: $\quad$ set $\mathcal{D} \leftarrow \mathcal{D} \cup \mathsf{tag}^{(2)}$;
10: $\quad \mathcal{R}_v^{(2)} \leftarrow \mathcal{O}_v(N_1, x_2, \mathsf{tag}^{(2)}, \lambda_1)$;
11: **while** $\mathcal{R}_v^{(2)} = \mathsf{no}$;
12: set $\mathcal{C}_1 \leftarrow \{\}$;
13: choose $x_3 \leftarrow \mathcal{M} \setminus \{x_1, x_2\}$;
14: **do**
15: $\quad$ choose $c_1 \leftarrow \{0,1\}^{n-\lambda_1} \setminus \mathcal{C}_1$;
16: $\quad$ set $\mathcal{C}_1 \leftarrow \mathcal{C}_1 \cup c_1$;
17: $\quad$ using Proposition 1 solve $\mathsf{Hash}_{\tau_{\lambda_1}}(x_1) \oplus \mathsf{Hash}_{\tau_{\lambda_1}}(x_2) = (\mathsf{tag}^{(1)} \oplus \mathsf{tag}^{(2)})||c_1$
18: $\quad\quad$ for τ_{λ_1} and let the solution be τ_{c_1};
19: $\quad$ set $x_{c_1} \leftarrow \mathsf{tag}^{(1)} \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_{c_1}}(x_1))$;
20: $\quad \mathcal{R}_v^{(3)} \leftarrow \mathcal{O}_v(N_1, x_3, x_{c_1} \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_{c_1}}(x_3)), \lambda_1)$;
21: **while** $\mathcal{R}_v^{(3)} = \mathsf{no}$;
22: choose any $x_4 \in \mathcal{M}$ and let
23: $\quad \mathsf{tag}^{(4)} \leftarrow \mathcal{O}_g(N_1, x_4, \lambda_2)$;
24: set $\mathcal{C}_2 \leftarrow \{\}$;
25: choose $x \in \mathcal{M} \setminus \{x_4\}$;
26: **do**
27: $\quad$ choose $c_2 \leftarrow \{0,1\}^{n-\lambda_1} \setminus \mathcal{C}_2$;
28: $\quad$ set $\mathcal{C}_2 \leftarrow \mathcal{C}_2 \cup c_2$;
29: $\quad$ solve $\mathsf{Hash}_{\tau_{\lambda_2}}(x_4) = \mathsf{msb}_{\lambda_1}(\mathsf{tag}^{(4)}) \oplus x_{c_1}||c_2$
30: $\quad\quad$ for τ_{λ_2} and let the solution be τ_{c_2};
31: $\quad$ set $x_{c_2} \leftarrow (\mathsf{msb}_{\lambda_1}(\mathsf{tag}^{(4)}) \oplus x_{c_1}||c_2) \oplus \mathsf{tag}^{(4)}$;
32: $\quad \mathcal{R}_v^{(5)} \leftarrow \mathcal{O}_v(N_1, x, x_{c_2} \oplus \mathsf{Hash}_{\tau_{c_2}}(x), \lambda_2)$;
33: **while** $\mathcal{R}_v^{(5)} = \mathsf{no}$.

Lemma 5. *The attack mentioned in Algorithm 2 forges with probability 1 and requires 2 tag generation queries and at most* $2^{\lambda_1} + 2 \times 2^{n-\lambda_1}$ *verification queries including the forgery attempts.*

Proof. That the attack mentioned in Algorithm 2 forges with probability 1 is proved if it can be shown that there is an iteration of the do-while loop in Steps 26 to 33 such that $\mathcal{R}_v^{(5)} = \mathsf{yes}$, i.e. there is a verification query in Step 32 which succeeds.
From Step 4, we get that

$$\mathsf{msb}_{\lambda_1}(\mathsf{F}_K(N_1) \oplus \mathsf{Hash}_{\tau_{\lambda_1}}(x_1)) = \mathsf{tag}^{(1)}. \tag{41}$$

As $\mathsf{NMAC\text{-}t6}(N_1, x_2, \lambda_1) \in \{0,1\}^{\lambda_1}$ and $\mathsf{tag}^{(2)}$ is varied through $\{0,1\}^{\lambda_1}$, the do-while loop in Steps 7 to 11 terminates and for the terminating $(N_1, x_2, \mathsf{tag}^{(2)}, \lambda_1)$ we get,

$$\mathsf{msb}_{\lambda_1}(\mathsf{F}_K(N_1) \oplus \mathsf{Hash}_{\tau_{\lambda_1}}(x_2)) = \mathsf{tag}^{(2)}. \tag{42}$$

So, from (41) and (42) we get,

$$\mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_{\lambda_1}}(x_1) \oplus \mathsf{Hash}_{\tau_{\lambda_1}}(x_2)) = \mathsf{tag}^{(1)} \oplus \mathsf{tag}^{(2)}. \tag{43}$$

Here $\mathsf{tag}^{(1)} \oplus \mathsf{tag}^{(2)}$ is a λ_1-bit binary string.
Following Proposition 1, for each choice of c_1 in the do-while loop in Steps 14 to 21, the equation in Step 17 can be solved to get τ_{c_1} and x_{c_1}.

The fact that $\mathsf{Hash}_{\tau_{\lambda_1}}(x_1) \oplus \mathsf{Hash}_{\tau_{\lambda_1}}(x_2) \in \{0,1\}^n$ and (43) suggest that there is a correct c_1, such that the equation in Step 17 holds and we consider that iteration of the do-while loop which deals with this particular c_1. The τ_{c_1} obtained in this iteration is the actual hash key used in the scheme. So,

$$\begin{aligned}
&\mathsf{NMAC\text{-}t6}(N_1, x_3, \lambda_1) \\
&= \mathsf{msb}_{\lambda_1}(\mathsf{F}_K(N_1) \oplus \mathsf{Hash}_{\tau_{c_1}}(x_3)) \\
&= \mathsf{tag}^{(1)} \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_{c_1}}(x_1)) \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_{c_1}}(x_3)) \qquad (44) \\
&= x_{c_1} \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_{c_1}}(x_3)). \qquad (45)
\end{aligned}$$

The expression in (44) comes from (41) and that in (45) comes from Step 19 in Algorithm 2.
Hence, in this particular iteration of the do-while loop, $\mathcal{R}_v^{(3)} = \mathsf{yes}$ and the loop terminates.
Next, from Step 23, we get

$$\mathsf{msb}_{\lambda_2}(\mathsf{F}_K(N_1) \oplus \mathsf{Hash}_{\tau_{\lambda_2}}(x_4)) = \mathsf{tag}^{(4)}.$$

As, $\lambda_2 = n$, so we can write,

$$\begin{aligned}
\mathsf{F}_K(N_1) \oplus \mathsf{Hash}_{\tau_{\lambda_2}}(x_4) &= \mathsf{tag}^{(4)} \\
\mathsf{Hash}_{\tau_{\lambda_2}}(x_4) &= \mathsf{tag}^{(4)} \oplus \mathsf{F}_K(N_1). \qquad (46)
\end{aligned}$$

Here, the n bits of $\mathsf{tag}^{(4)}$ and $\mathsf{msb}_{\lambda_1}(\cdot)$ of $\mathsf{F}_K(N_1)$, which is x_{c_1}, are known.

As $\mathsf{Hash}_{\tau_{\lambda_2}}(x_4) \in \{0,1\}^n$, there is a $c_2 \in \{0,1\}^{n-\lambda_1}$, such that,

$$\mathsf{Hash}_{\tau_{\lambda_2}}(x_4) = \mathsf{msb}_{\lambda_1}(\mathsf{tag}^{(4)} \oplus \mathsf{F}_K(N_1))||c_2 = (\mathsf{msb}_{\lambda_1}(\mathsf{tag}^{(4)}) \oplus x_{c_1})||c_2. \tag{47}$$

Now, using a suitable length x_4, for the correct c_2, the actual values of τ_{c_2} and x_{c_2} are obtained in Steps 30 and 31 respectively.
For the correct c_2, from (46) and (47), we get,

$$\mathsf{F}_K(N_1) = \mathsf{Hash}_{\tau_{\lambda_2}}(x_4) \oplus \mathsf{tag}^{(4)} = ((\mathsf{msb}_{\lambda_1}(\mathsf{tag}^{(4)}) \oplus x_{c_1})||c_2) \oplus \mathsf{tag}^{(4)}, \tag{48}$$

which equals x_{c_2} according to Step 31 in Algorithm 2.
Hence, for the $x \in \mathcal{M} \setminus \{x_4\}$,

$$\begin{aligned}
&\mathsf{NMAC\text{-}t6}(N_1, x, \lambda_2) \\
&= \mathsf{F}_K(N_1) \oplus \mathsf{Hash}_{\tau_{c_2}}(x) \\
&= x_{c_2} \oplus \mathsf{Hash}_{\tau_{c_2}}(x). \qquad (49)
\end{aligned}$$

The last equality follows from (48).
From (49), it is clear that for the iteration of the do-while loop in Steps 26 to 33, in which the correct c_2 is used, $\mathcal{R}_v^{(5)} = \mathsf{yes}$ with probability 1, which proves the first part of the Lemma.

In the attack, there are 2 tag generation queries in Steps 4 and 23. In each iteration of each of the do-while loops in Steps 7 to 11, in 14 to 21 and in 26 to 33 one verification query is made. The do-while loop in Steps 7 to 11 iterates at most 2^{λ_1} times for different values of $\mathsf{tag}^{(2)}$; the do-while loop in Steps 14 to 21 iterates at most $2^{n-\lambda_1}$ times for different values of c_1 and that in Steps 26 to 33 iterates at most $2^{n-\lambda_1}$ times for different values of c_2, as $\mathsf{tag}^{(2)}$ is λ_1-bit quantity and each of c_1 and c_2 is $(n-\lambda_1)$-bit quantity. Hence, the attack requires 2 tag generation queries and at most $2^{\lambda_1} + 2 \times 2^{n-\lambda_1}$ verification queries including the forgery. □

B Security of the PRF Schemes in Table 6

Here, we provide the security proofs of the PRF schemes in Table 6. For the proofs we require the following additional notion for a hash function $\{H_\tau\}_{\tau\in\Theta}$, where for each $\tau \in \Theta$, $H_\tau : \mathcal{M} \to \{0,1\}^n$.

- For distinct $x, x' \in \mathcal{M}$, the **collision probability** of $\{H_\tau\}_{\tau\in\Theta}$, for the pair (x, x') is defined to be $\Pr_\tau[H_\tau(x) = H_\tau(x')]$. Here the probability is taken over the uniform random choice of τ from Θ.
  Similar to differential probability, the collision probability may also depend on the lengths of x and x'. As mentioned earlier, L is the maximum of the lengths of the binary strings in $\mathcal{M}$. Let $\varepsilon : \{0,\ldots,L\}^2 \to [0,1]$ be a function such that the collision probability for any (x, x') is at most $\varepsilon(\mathsf{len}(x), \mathsf{len}(x'))$. Then the family $\{H_\tau\}_{\tau\in\Theta}$ is said to be ε-**AU**.

- Let $\{H_\tau\}_{\tau\in\Theta}$ satisfy the following properties.

  - For τ_1 and τ_2 chosen independently and uniformly at random from Θ and for x_1 and x_2 (not necessarily distinct) from $\mathcal{M}$,

    $$\Pr[H_{\tau_1}(x_1) = H_{\tau_2}(x_2)] \leq \varepsilon_1(\mathsf{len}(x_1), \mathsf{len}(x_2)), \tag{50}$$

    where $\varepsilon_1 : \{0,\ldots,L\}^2 \to [0,1]$.
  - For any $a \in \{0,1\}^n$, an $x \in \mathcal{M}$ and τ chosen uniformly at random from Θ,

    $$\Pr[H_\tau(x) = a] \leq \varepsilon_2(\mathsf{len}(x)), \tag{51}$$

    where $\varepsilon_2 : \{0,\ldots,L\} \to [0,1]$.
    Then the family $\{H_\tau\}_{\tau\in\Theta}$ is said to be $(\varepsilon_1, \varepsilon_2)$-**eligible**.

- As mentioned earlier in the context of nonce-based MACs, depending upon the actual choice of the hash function, the key could either be a fixed length string, or, it could be a variable length string which depends upon the length of the message. In either case, we will consider the key space Θ to consist of fixed length keys, where this length of the keys is some sufficiently large fixed value, such that any practical size message can be taken care of by using the appropriate length prefix of the key.

Theorem 6. *Let $q \geq 1$. Fix the query profile $\mathfrak{C} = ((x^{(1)}, \lambda^{(1)}), \ldots, (x^{(q)}, \lambda^{(q)}))$. Let the total number of bits in the queries be $\sigma \geq 1$. In* F-PRF *defined in Table 6, suppose that the hash function $\{\mathsf{Hash}_\tau\}_{\tau\in\Theta}$ is ε-AU, where $\varepsilon(\cdot,\cdot) \geq 1/2^n$. Let, $l^{(i)} = \mathsf{len}(\mathsf{bin}_8(\lambda^{(i)})||x^{(i)})$, where $1 \leq i \leq q$.*

Then

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F\text{-}PRF}}(t,q,\sigma) \;\leq\; \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(t+t',q) + \sum_{1\leq i<j\leq q} \varepsilon(l^{(i)},l^{(j)}) \tag{52}$$

where t' is the time required to hash q pairs of message and tag length of total length at most σ, plus some bookkeeping time.

Proof. Let $\mathcal{A}$ be the adversary attacking the PRF-security of $\mathsf{F\text{-}PRF}$. From $\mathcal{A}$, we define an adversary $\mathcal{B}$ attacking the PRF-security of F. $\mathcal{B}$ has access to an oracle, which is either the real oracle (i.e. $\mathsf{F}_K(\cdot)$ instantiated with an independent and uniform random K from $\mathcal{K}$) or the random oracle which on provided with distinct n-bit inputs, outputs independent and uniform random n-bit strings. $\mathcal{B}$ simulates $\mathcal{A}$ in the following manner. $\mathcal{B}$ maintains a list $\mathfrak{L}$ of tuples of the form (N,R) which is initially empty. At the outset, it chooses an independent and uniform random τ from Θ. For each query $(x^{(i)},\lambda^{(i)})$ made by $\mathcal{A}$, $\mathcal{B}$ uses this τ to hash $(\mathsf{bin}_8(\lambda^{(i)})||x^{(i)})$ and obtains the digest N. Now, it searches for a tuple containing N in $\mathfrak{L}$. If no such tuple is present, $\mathcal{B}$ queries its oracle on the input N to get the output corresponding to the call to F. Let the output, either obtained from $\mathfrak{L}$ or from the oracle, be R. If it is obtained from the oracle, i.e. it is not present in $\mathfrak{L}$ already, $\mathcal{B}$ adds the tuple (N,R) to the list and sends R to $\mathcal{A}$. At the end, $\mathcal{B}$ outputs whatever bit $\mathcal{A}$ outputs. By using $\mathfrak{L}$, it is ensured that $\mathcal{B}$ queries its oracle only on distinct inputs for any combination of queries made by $\mathcal{A}$ and $\mathcal{B}$ makes at most $q'\leq q$ oracle queries.

By $\mathcal{B}^{\mathsf{real}}\Rightarrow 1$ (resp. $\mathcal{B}^{\mathsf{rnd}}\Rightarrow 1$) we denote the event that $\mathcal{B}$ outputs the bit 1 after interacting with the real (resp. random) oracle. By definition

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{B}) \;=\; \Pr[\mathcal{B}^{\mathsf{real}}\Rightarrow 1] - \Pr[\mathcal{B}^{\mathsf{rnd}}\Rightarrow 1]. \tag{53}$$

If $\mathcal{B}$'s oracle is real, then the following is immediate.

$$\Pr[\mathcal{A}^{\mathsf{real}}\Rightarrow 1] = \Pr[\mathcal{B}^{\mathsf{real}}\Rightarrow 1]. \tag{54}$$

We now consider the situation when $\mathcal{B}$'s oracle is random. In the subsequent discussion, the numbers in the superscripts of the variables denote the sequence of the queries.

The q queries made by $\mathcal{A}$ are $(x^{(i)},\lambda^{(i)})$, $i=1,\ldots,q$. Define

$$\mathsf{Dom} = \{N^{(1)},\ldots,N^{(q)}\}$$

and let Bad be the event that two variables in Dom have the same value.

The variable $N^{(i)}$ is obtained as $N^{(i)} = \mathsf{Hash}_\tau(\mathsf{bin}_8(\lambda^{(i)})||x^{(i)})$, $i=1,\ldots,q$. So, from the fact that $\{\mathsf{Hash}_\tau\}$ is ε-AU, we get the probability that $N^{(i)}=N^{(j)}$ is at most $\varepsilon(l^{(i)},l^{(j)})$ and as a result,

$$\Pr[\mathsf{Bad}] \leq \sum_{1\leq i<j\leq q} \varepsilon(l^{(i)},l^{(j)}) \tag{55}$$

Further,

$$\Pr[\mathcal{B}^{\mathsf{rnd}}\Rightarrow 1] \;\leq\; \Pr[\mathcal{B}^{\mathsf{rnd}}\Rightarrow 1\wedge\overline{\mathsf{Bad}}] + \Pr[\mathsf{Bad}]. \tag{56}$$

Seen in conjunction with $\overline{\mathsf{Bad}}$, the outputs of the random oracle on the inputs $N^{(i)}$, $i=1,\ldots,q$ are independent and uniform random n-bit strings. So, we have

$$\Pr[\mathcal{A}^{\mathsf{rnd}}\Rightarrow 1\wedge\overline{\mathsf{Bad}}] \;=\; \Pr[\mathcal{B}^{\mathsf{rnd}}\Rightarrow 1\wedge\overline{\mathsf{Bad}}]. \tag{57}$$

Combining (53), (54), (56) and (57) we obtain:

$$\begin{aligned}
\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{B}) &= \Pr[\mathcal{B}^{\mathsf{real}} \Rightarrow 1] - \Pr[\mathcal{B}^{\mathsf{rnd}} \Rightarrow 1] \\
&= \Pr[\mathcal{A}^{\mathsf{real}} \Rightarrow 1] - \Pr[\mathcal{B}^{\mathsf{rnd}} \Rightarrow 1] \\
&\geq \Pr[\mathcal{A}^{\mathsf{real}} \Rightarrow 1] - \Big(\Pr[\mathcal{B}^{\mathsf{rnd}} \Rightarrow 1 \wedge \overline{\mathsf{Bad}}] + \Pr[\mathsf{Bad}]\Big) \\
&= \Pr[\mathcal{A}^{\mathsf{real}} \Rightarrow 1] - \Big(\Pr[\mathcal{A}^{\mathsf{rnd}} \Rightarrow 1 \wedge \overline{\mathsf{Bad}}] + \Pr[\mathsf{Bad}]\Big) \\
&= \Pr[\mathcal{A}^{\mathsf{real}} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathsf{rnd}} \Rightarrow 1 \wedge \overline{\mathsf{Bad}}] - \Pr[\mathsf{Bad}] \\
&\geq \Pr[\mathcal{A}^{\mathsf{real}} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathsf{rnd}} \Rightarrow 1] - \Pr[\mathsf{Bad}] \\
&= \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F\text{-}PRF}}(\mathcal{A}) - \Pr[\mathsf{Bad}]. \qquad (58)
\end{aligned}$$

Rearranging (58) gives

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F\text{-}PRF}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{B}) + \Pr[\mathsf{Bad}]. \qquad (59)$$

Putting value of $\Pr[\mathsf{Bad}]$ from (55) gives

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F\text{-}PRF}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{B}) + \sum_{1 \leq i < j \leq q} \varepsilon(l^{(i)}, l^{(j)}). \qquad (60)$$

The resource bounded advantage follows from this on noting that the number of queries made by $\mathcal{B}$ is upper bounded by the number of queries made by $\mathcal{A}$ and for each query made by $\mathcal{A}$, $\mathcal{B}$ has to hash the message and tag-length pair. □

Theorem 7. *Let $q \geq 1$. Fix the query profile $\mathfrak{C} = ((x^{(1)}, \lambda^{(1)}), \ldots, (x^{(q)}, \lambda^{(q)}))$. Let the total number of bits in the queries be $\sigma \geq 1$. In* SC-PRF *defined in Table 6, suppose that the hash function $\{\mathsf{Hash}_\tau\}_{\tau \in \Theta}$ is ε-AU, where $\varepsilon(\cdot,\cdot) \geq 1/2^n$. Further, suppose that Hash_τ is $(\varepsilon, \varepsilon')$-eligible hash function, where $\varepsilon'(\cdot) \geq 1/2^n$. Let, $l^{(i)} = \mathsf{len}(x^{(i)})$, where $1 \leq i \leq q$.*

Then

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC\text{-}PRF}}(t, q, \sigma) \leq \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(t + t', 2q) + \sum_{1 \leq i < j \leq q} \varepsilon(l^{(i)}, l^{(j)}) + \min(q, \#\mathcal{L}) \times \sum_{1 \leq k \leq q} \varepsilon'(l^{(k)}). \qquad (61)$$

Here, t' is the time required to calculate the hash of q messages of total length at most σ, plus some basic computation and bookkeeping time.

Proof. Let as in the proof of Theorem 6, $\mathcal{A}$ be the adversary attacking the PRF-security of SC-PRF. From $\mathcal{A}$, we define an adversary $\mathcal{B}$ attacking the PRF-security of SC. $\mathcal{B}$ has access to an oracle, which is either the real oracle (i.e. $\mathsf{SC}_K(\cdot)$ instantiated with an independent and uniform random K from $\mathcal{K}$) or the random oracle which on provided with distinct n-bit inputs, outputs independent and uniform random L-bit strings. $\mathcal{B}$ simulates $\mathcal{A}$ in the following manner.

$\mathcal{B}$ maintains a list $\mathfrak{L}$ of tuples of the form $(\mathsf{inp}, \mathsf{out})$ which is initially empty. Here, for each tuple, out is L-bit long. For any query $(x^{(i)}, \lambda^{(i)})$ made by $\mathcal{A}$, $\mathcal{B}$ first forms str as $\mathsf{str} = \mathsf{bin}_8(\lambda^{(i)})||\mathsf{fStr}$ and calculates $b = \mathfrak{b}(x^{(i)})$. Now, it searches for a tuple containing str in $\mathfrak{L}$. If no such tuple is present, $\mathcal{B}$ queries its oracle on the input str to get the output corresponding to the call to SC. Let the output, either obtained from $\mathfrak{L}$ or from the oracle, be τ. If it is obtained from the oracle, i.e. it is not present in $\mathfrak{L}$ already, $\mathcal{B}$ adds the tuple (str, τ) to the list.

Now, it uses τ (or the appropriate length prefix of τ according to b) as the hash key to compute the hash of $x^{(i)}$. Let, the digest be N. Again, $\mathcal{B}$ searches for a tuple containing N in $\mathfrak{L}$. If no such tuple is present, $\mathcal{B}$ queries its oracle on the input N to get the output corresponding to the call to SC. This time let the output, either obtained from $\mathfrak{L}$ or from the oracle, be R. If it is obtained from the oracle, i.e. it is not present in $\mathfrak{L}$ already, $\mathcal{B}$ adds the tuple (N, R) to the list and sends R to $\mathcal{A}$. At the end, $\mathcal{B}$ outputs whatever bit $\mathcal{A}$ outputs. By using $\mathfrak{L}$, it is ensured that $\mathcal{B}$ queries its oracle only on distinct inputs for any combination of queries made by $\mathcal{A}$ and $\mathcal{B}$ makes at most $2q' \leq 2q$ oracle queries.

By $\mathcal{B}^{\mathsf{real}} \Rightarrow 1$ (resp. $\mathcal{B}^{\mathsf{rnd}} \Rightarrow 1$) we denote the event that $\mathcal{B}$ outputs the bit 1 after interacting with the real (resp. random) oracle. By definition

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(\mathcal{B}) \quad = \quad \Pr[\mathcal{B}^{\mathsf{real}} \Rightarrow 1] - \Pr[\mathcal{B}^{\mathsf{rnd}} \Rightarrow 1]. \tag{62}$$

If $\mathcal{B}$'s oracle is real, then the following is immediate.

$$\Pr[\mathcal{A}^{\mathsf{real}} \Rightarrow 1] = \Pr[\mathcal{B}^{\mathsf{real}} \Rightarrow 1]. \tag{63}$$

We now consider the situation when $\mathcal{B}$'s oracle is random. This is essentially the part of the argument which differs from that for Theorem 6. Here also, the numbers in the superscripts of the variables denote the sequence of the queries.

We define Dom_1 and Dom_2 as in Table 7.

Table 7: Collision analysis.

$\mathsf{Dom}_1 = \{N^{(1)}, \ldots, N^{(q)}\};$

Subroutine $\mathsf{CreateDom}_2(X)$
$\mathsf{Dom}_2 \leftarrow \{\};$
for $i = 1$ to q
if $\mathsf{bin}_8(\lambda^{(i)})\|\|\mathsf{fStr} \notin \mathsf{Dom}_2$ then
$\mathsf{Dom}_2 \leftarrow \mathsf{Dom}_2 \cup \{\mathsf{bin}_8(\lambda^{(i)})\|\|\mathsf{fStr}\};$
endif;
endfor;

Let Bad_1 be the event that two random variables in Dom_1 have the same value; Bad_2 be the event that some random variable in Dom_1 is equal to some random variable in Dom_2; and let $\mathsf{Bad} = \mathsf{Bad}_1 \vee \mathsf{Bad}_2$. The number of elements in Dom_2 is upper bounded by the minimum of q and the number of elements in $\mathcal{L}$.

The q queries made by $\mathcal{A}$ are $(x^{(i)}, \lambda^{(i)})$, $i = 1, \ldots, q$. By the restriction on an adversary that it cannot repeat queries, these pairs are distinct. For $1 \leq i, j \leq q$, if $\lambda^{(i)} \neq \lambda^{(j)}$, then $\tau^{(i)}$ and $\tau^{(j)}$ are independent and uniform random. Hence, as $\{\mathsf{Hash}_\tau\}$ is $(\varepsilon, \varepsilon')$-eligible hash function, it turns out that the probability that $N^{(i)} = N^{(j)}$ is at most $\varepsilon(l^{(i)}, l^{(j)})$. On the other hand, if $\lambda^{(i)} = \lambda^{(j)}$, then definitely $x^{(i)} \neq x^{(j)}$ and hence from the fact that $\{\mathsf{Hash}_\tau\}$ is ε-AU, again the probability that $N^{(i)} = N^{(j)}$ is at most $\varepsilon(l^{(i)}, l^{(j)})$. Hence,

$$\Pr[\mathsf{Bad}_1] \quad \leq \quad \textstyle\sum_{1 \leq i < j \leq q} \varepsilon(l^{(i)}, l^{(j)}); \tag{64}$$

The random variables in Dom_1 are of the form $N^{(k)} = \mathsf{Hash}_{\tau^{(k)}}(x^{(k)})$ for $1 \leq k \leq q$, where $\tau^{(k)}$ is uniform random and $x^{(k)} \in \mathcal{M}$. On the other hand, the random variables in Dom_2 are n-bit strings. From the fact that the hash function is $(\varepsilon, \varepsilon')$-eligible, we get, for any $1 \leq k \leq q$ and any $a \in \mathsf{Dom}_2$, $\Pr[N^{(k)} = a] \leq \varepsilon'(l^{(k)})$ and hence,

$$\Pr[\mathsf{Bad}_2] \;\leq\; \textstyle\sum_{1 \leq k \leq q} (\min(q, \#\mathcal{L}) \times \varepsilon'(l^{(k)})) \leq \min(q, \#\mathcal{L}) \times \sum_{1 \leq k \leq q} \varepsilon'(l^{(k)}); \tag{65}$$

So, from (64) and (65), we get,

$$\Pr[\mathsf{Bad}] \;\leq\; \textstyle\sum_{1 \leq i < j \leq q} \varepsilon(l^{(i)}, l^{(j)}) + \min(q, \#\mathcal{L}) \times \sum_{1 \leq k \leq q} \varepsilon'(l^{(k)}). \tag{66}$$

Again,

$$\Pr[\mathcal{B}^{\mathsf{rnd}} \Rightarrow 1] \;\leq\; \Pr[\mathcal{B}^{\mathsf{rnd}} \Rightarrow 1 \wedge \overline{\mathsf{Bad}}] + \Pr[\mathsf{Bad}]. \tag{67}$$

Now, seen in conjunction with $\overline{\mathsf{Bad}}$, the random variables in Dom_1 take distinct values and these values are also distinct from the values of the random variables in Dom_2. As a result, the outputs of the random oracle on the inputs from Dom_1 are independent and uniform random strings. So, we have

$$\Pr[\mathcal{A}^{\mathsf{rnd}} \Rightarrow 1 \wedge \overline{\mathsf{Bad}}] \;=\; \Pr[\mathcal{B}^{\mathsf{rnd}} \Rightarrow 1 \wedge \overline{\mathsf{Bad}}]. \tag{68}$$

Hence, as in (59) obtained as part of the proof of Theorem 6, here we get,

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC\text{-}PRF}}(\mathcal{A}) \;\leq\; \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(\mathcal{B}) + \Pr[\mathsf{Bad}]. \tag{69}$$

Using the upper bound on $\Pr[\mathsf{Bad}]$ from (66) gives,

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC\text{-}PRF}}(\mathcal{A}) \;\leq\; \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(\mathcal{B}) + \sum_{1 \leq i < j \leq q} \varepsilon(l^{(i)}, l^{(j)}) + \min(q, \#\mathcal{L}) \times \sum_{1 \leq k \leq q} \varepsilon'(l^{(k)}). \tag{70}$$

The resource bounded advantage follows from this on noting that the number of queries made by $\mathcal{B}$ is upper bounded by twice the number of queries made by $\mathcal{A}$ and for each query made by $\mathcal{A}$, $\mathcal{B}$ has to calculate the function $\mathfrak{b}(\cdot)$, hash the message and do some basic computation and bookkeeping. $\square$

Theorem 8. *Let $q \geq 1$. Fix the query profile $\mathfrak{C} = ((x^{(1)}, \lambda^{(1)}), \ldots, (x^{(q)}, \lambda^{(q)}))$. Let the total number of bits in the queries be $\sigma \geq 1$. In* E-PRF *defined in Table 6, suppose that the hash function $\{\mathsf{Hash}_\tau\}_{\tau \in \Theta}$ is ε-AU, where $\varepsilon(\cdot, \cdot) \geq 1/2^n$. Further, suppose that Hash_τ is $(\varepsilon, \varepsilon')$-eligible hash function, where $\varepsilon'(\cdot) \geq 1/2^n$. Let $l^{(i)} = \mathsf{len}(x^{(i)})$ and $b^{(i)} = \mathfrak{b}(x^{(i)})$, where $1 \leq i \leq q$. Let $\mathsf{nBlks} = \sum_{i=1}^{q} b^{(i)}$.*

Then

$$\begin{aligned} \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{E\text{-}PRF}}(t, q, \sigma) \;&\leq\; \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{E}}(t + t', 2q + \mathsf{nBlks}) + \sum_{1 \leq i < j \leq q} \varepsilon(l^{(i)}, l^{(j)}) \\ &+\; \frac{1}{2^n} \times \sum_{1 \leq i < j \leq q} (b^{(i)} \times b^{(j)}) + \min(q, \#\mathcal{L}) \times \sum_{1 \leq k \leq q} \varepsilon'(l^{(k)}) + \frac{q \times \mathsf{nBlks}}{2^n}. \end{aligned} \tag{71}$$

Here, t' is the time required to calculate the hash of q messages of total length at most σ, plus some basic computation and bookkeeping time.

Proof. In the subsequent discussion, the numbers in the superscripts of the variables denote the sequence of the queries.

Let as above, $\mathcal{A}$ be the adversary attacking the PRF-security of E-PRF. From $\mathcal{A}$, we define an adversary $\mathcal{B}$ attacking the PRF-security of E. $\mathcal{B}$ has access to an oracle, which is either the real oracle (i.e. $\mathsf{E}_K(\cdot)$ instantiated with an independent and uniform random K from $\mathcal{K}$) or the random oracle which on provided with distinct n-bit inputs, outputs independent and uniform random n-bit strings. $\mathcal{B}$ simulates $\mathcal{A}$ in the following manner. $\mathcal{B}$ maintains a list $\mathfrak{L}$ of tuples of the form $(\mathsf{inp}, \mathsf{out})$ which is initially empty. For any query $(x^{(i)}, \lambda^{(i)})$ made by $\mathcal{A}$, $\mathcal{B}$ first forms $\mathsf{str}^{(i)}$ as $\mathsf{str}^{(i)} = \mathsf{bin}_8(\lambda^{(i)})||\mathsf{fStr}$ and computes $b^{(i)} = \mathfrak{b}(x^{(i)})$. Now, it searches for a tuple containing $\mathsf{str}^{(i)}$ in $\mathfrak{L}$. If no such tuple is present, $\mathcal{B}$ queries its oracle on the input $\mathsf{str}^{(i)}$ to get the output corresponding to the call to E. Let the output, either obtained from $\mathfrak{L}$ or from the oracle, be $\mathsf{Q}^{(i)}$. If it is obtained from the oracle, i.e. it is not present in $\mathfrak{L}$ already, $\mathcal{B}$ adds the tuple $(\mathsf{str}^{(i)}, \mathsf{Q}^{(i)})$ to the list.

Now, if $b^{(i)} > 1$, then $\mathcal{B}$ searches, in $\mathfrak{L}$, for tuples containing the oracle outputs corresponding to $\mathsf{Q}^{(i)} \oplus \mathsf{bin}_n(1)$, $\mathsf{Q}^{(i)} \oplus \mathsf{bin}_n(2)$, $\cdots$, $\mathsf{Q}^{(i)} \oplus \mathsf{bin}_n(b^{(i)})$. If at least one of these is not present, $\mathcal{B}$ queries its oracle to get the corresponding outputs for those which are not present in the list and adds these new tuples to it. Now, it forms τ by concatenating these outputs if $b^{(i)} > 1$; otherwise $\tau = \mathsf{Q}^{(i)}$. This τ is used to compute the hash of $x^{(i)}$. Let the digest be $N^{(i)}$.

Again, it searches for a tuple containing $N^{(i)}$ in $\mathfrak{L}$. If no such tuple is present, $\mathcal{B}$ queries its oracle on the input $N^{(i)}$ to get the output. Let the output, either obtained from $\mathfrak{L}$ or from the oracle, be $R^{(i)}$. If it is obtained from the oracle, i.e. it is not present in $\mathfrak{L}$ already, $\mathcal{B}$ adds the tuple $(N^{(i)}, R^{(i)})$ to the list and sends $R^{(i)}$ to $\mathcal{A}$. At the end, $\mathcal{B}$ outputs whatever bit $\mathcal{A}$ outputs. By using $\mathfrak{L}$, it is ensured that $\mathcal{B}$ queries its oracle only on distinct inputs for any combination of queries made by $\mathcal{A}$. Here, $\mathcal{B}$ makes at most $2q + \sum_{i=1}^{q} b^{(i)} = 2q + \mathsf{nBlks}$ oracle queries. Note that it has to make extra $b^{(i)}$ oracle queries for the i-th query if and only if the hash key corresponding to this query contains more than one n-bit block, i.e. $b^{(i)} > 1$.

By $\mathcal{B}^{\mathsf{real}} \Rightarrow 1$ (resp. $\mathcal{B}^{\mathsf{rnd}} \Rightarrow 1$) we denote the event that $\mathcal{B}$ outputs the bit 1 after interacting with the real (resp. random) oracle. By definition

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{E}}(\mathcal{B}) \quad = \quad \Pr[\mathcal{B}^{\mathsf{real}} \Rightarrow 1] - \Pr[\mathcal{B}^{\mathsf{rnd}} \Rightarrow 1]. \tag{72}$$

If $\mathcal{B}$'s oracle is real, then the following is immediate.

$$\Pr[\mathcal{A}^{\mathsf{real}} \Rightarrow 1] = \Pr[\mathcal{B}^{\mathsf{real}} \Rightarrow 1]. \tag{73}$$

We now consider the situation when $\mathcal{B}$'s oracle is random. This part is somewhat different from that of Theorem 7.

Define Dom_1, Dom_2 and Dom_3 as in Table 8.

Let Bad_1 be the event that two random variables in Dom_1 have the same value; Bad_2 be the event that some random variable in Dom_1 is equal to some random variable in Dom_2; Bad_3 be the event that some random variable in Dom_1 is equal to some random variable in Dom_3; and let $\mathsf{Bad} = \mathsf{Bad}_1 \vee \mathsf{Bad}_2 \vee \mathsf{Bad}_3$. The number of elements in Dom_2 is at most the minimum of q and the number of elements in $\mathcal{L}$.

Note that, corresponding to the i-th query, $1 \leq i \leq q$, Dom_3 is non-empty if and only if $b^{(i)} > 1$. In that case, Bad_3 is a null event and $\mathsf{Bad} = \mathsf{Bad}_1 \vee \mathsf{Bad}_2$. The number of elements in Dom_3 is at most $\sum_{i=1}^{q} b^{(i)} = \mathsf{nBlks}$.

For $1 \leq i, j \leq q$, there are two cases.

Table 8: Collision analysis.

$\mathsf{Dom}_1 = \{N^{(1)}, \ldots, N^{(q)}\}$;

Subroutine $\mathsf{CreateDom}_2(X)$
    $\mathsf{Dom}_2 \leftarrow \{\}$;
    for $i = 1$ to q
        if $\mathsf{bin}_8(\lambda^{(i)})||\mathsf{fStr} \notin \mathsf{Dom}_2$ then
            $\mathsf{Dom}_2 \leftarrow \mathsf{Dom}_2 \cup \{\mathsf{bin}_8(\lambda^{(i)})||\mathsf{fStr}\}$;
        endif;
    endfor;

Subroutine $\mathsf{CreateDom}_3(X)$
    $\mathsf{Dom}_3 \leftarrow \{\}$;
    for $i = 1$ to q
        if $b^{(i)} > 1$
        for $j = 1$ to $b^{(i)}$
            if $\mathsf{Q}^{(i)} \oplus \mathsf{bin}_n(j) \notin \mathsf{Dom}_3$ then
                $\mathsf{Dom}_3 \leftarrow \mathsf{Dom}_3 \cup \{\mathsf{Q}^{(i)} \oplus \mathsf{bin}_n(j)\}$;
            endif;
        endfor;
        endif;
    endfor;

- Case $\lambda^{(i)} \neq \lambda^{(j)}$: In this case, $\mathsf{Q}^{(i)}$ and $\mathsf{Q}^{(j)}$ are independent and uniform random. There are three subcases.
  - Subcase: $b^{(i)} = b^{(j)} = 1$. In this case,
$$\begin{aligned} \Pr[N^{(i)} = N^{(j)}] &= \Pr[\mathsf{Hash}_{\mathsf{Q}^{(i)}}(x^{(i)}) = \mathsf{Hash}_{\mathsf{Q}^{(j)}}(x^{(j)})] \\ &\leq \varepsilon(l^{(i)}, l^{(j)}). \end{aligned} \tag{74}$$
    The last inequality follows from the fact that Hash is an $(\varepsilon, \varepsilon')$-eligible hash function.
  - Subcase: Exactly one of $b^{(i)}$ and $b^{(j)}$ is more than 1. We consider without loss of generality that $b^{(i)} = 1$ and $b^{(j)} > 1$; in this case, consider the set
$$\mathfrak{D}_j = \{\mathsf{Q}^{(j)} \oplus \mathsf{bin}_n(1), \mathsf{Q}^{(j)} \oplus \mathsf{bin}_n(2), \cdots, \mathsf{Q}^{(j)} \oplus \mathsf{bin}_n(b^{(j)})\}.$$
    The key $\tau^{(i)}$ (used to hash $x^{(i)}$) and the key $\tau^{(j)}$ (used to hash $x^{(j)}$) are independent if and only if $\mathsf{bin}_8(\lambda^{(i)})||\mathsf{fStr} \notin \mathfrak{D}_j$. Hence,
$$\begin{aligned} \Pr[N^{(i)} = N^{(j)}] &= \Pr[\mathsf{Hash}_{\tau^{(i)}}(x^{(i)}) = \mathsf{Hash}_{\tau^{(j)}}(x^{(j)})] \\ &\leq \Pr[\mathsf{Hash}_{\tau^{(i)}}(x^{(i)}) = \mathsf{Hash}_{\tau^{(j)}}(x^{(j)}) | \mathsf{bin}_8(\lambda^{(i)})||\mathsf{fStr} \notin \mathfrak{D}_j] \\ &\quad + \Pr[\mathsf{bin}_8(\lambda^{(i)})||\mathsf{fStr} \in \mathfrak{D}_j] \\ &\leq \varepsilon(l^{(i)}, l^{(j)}) + \frac{1}{2^n} \times b^{(j)}. \end{aligned} \tag{75}$$
    The last inequality follows from the fact that Hash is $(\varepsilon, \varepsilon')$-eligible hash function and $\mathsf{Q}^{(j)}$ is uniformly distributed random variable.

– Subcase: $b^{(i)} > 1$ and $b^{(j)} > 1$. Consider the two sets

$$\begin{aligned}\mathfrak{D}_i &= \{\mathsf{Q}^{(i)} \oplus \mathsf{bin}_n(1), \mathsf{Q}^{(i)} \oplus \mathsf{bin}_n(2), \cdots, \mathsf{Q}^{(i)} \oplus \mathsf{bin}_n(b^{(i)})\};\\ \mathfrak{D}_j &= \{\mathsf{Q}^{(j)} \oplus \mathsf{bin}_n(1), \mathsf{Q}^{(j)} \oplus \mathsf{bin}_n(2), \cdots, \mathsf{Q}^{(j)} \oplus \mathsf{bin}_n(b^{(j)})\}.\end{aligned}$$

Now, $\tau^{(i)}$ and $\tau^{(j)}$ are independent if and only if no random variable in $\mathfrak{D}_i$ is equal to any random variable in $\mathfrak{D}_j$. Let $\mathsf{Coll}(\mathfrak{D}_i, \mathfrak{D}_j)$ be the event that some random variable in $\mathfrak{D}_i$ is equal to some random variable in $\mathfrak{D}_j$.
So, in this case,

$$\begin{aligned}\Pr[N^{(i)} = N^{(j)}] &= \Pr[\mathsf{Hash}_{\tau^{(i)}}(x^{(i)}) = \mathsf{Hash}_{\tau^{(j)}}(x^{(j)})]\\ &\leq \Pr[\mathsf{Hash}_{\tau^{(i)}}(x^{(i)}) = \mathsf{Hash}_{\tau^{(j)}}(x^{(j)}) | \overline{\mathsf{Coll}(\mathfrak{D}_i, \mathfrak{D}_j)}]\\ &\quad + \Pr[\mathsf{Coll}(\mathfrak{D}_i, \mathfrak{D}_j)]\\ &\leq \varepsilon(l^{(i)}, l^{(j)}) + \frac{1}{2^n} \times b^{(i)} \times b^{(j)}. \qquad (76)\end{aligned}$$

The last inequality follows from the fact that Hash is $(\varepsilon, \varepsilon')$-eligible hash function and $\mathsf{Q}^{(i)}$ and $\mathsf{Q}^{(j)}$ are independent and uniform random.

- Case $\lambda^{(i)} = \lambda^{(j)}$: In this case, definitely $x^{(i)} \neq x^{(j)}$. If $(b^{(i)} = b^{(j)} = 1)$ or $(b^{(i)} > 1$ and $b^{(j)} > 1)$, then from the fact that Hash is ε-AU, we get $\Pr[N^{(i)} = N^{(j)}] \leq \varepsilon(l^{(i)}, l^{(j)})$. Note that, in both cases $\tau^{(i)} = \tau^{(j)}$ (in our setting, when $b^{(i)} > 1$ and $b^{(j)} > 1$, we have $\tau^{(i)} = \tau^{(j)}$ and for the hashing we use the appropriate length prefix of the key).
In the case when exactly one of $b^{(i)}$ and $b^{(j)}$ is greater than 1, i.e. $b^{(i)} = 1$ and $b^{(j)} > 1$ or vice versa, without loss of generality we take $b^{(i)} = 1$ and $b^{(j)} > 1$ and consider the set

$$\mathfrak{D}_j = \{\mathsf{Q}^{(j)} \oplus \mathsf{bin}_n(1), \mathsf{Q}^{(j)} \oplus \mathsf{bin}_n(2), \cdots, \mathsf{Q}^{(j)} \oplus \mathsf{bin}_n(b^{(j)})\}.$$

Again, $\tau^{(i)}$ and $\tau^{(j)}$ are independent of each other if and only if $\mathsf{bin}_8(\lambda^{(i)})||\mathsf{fStr} \notin \mathfrak{D}_j$. Hence, the bound stated in (75) applies here as well and we get

$$\Pr[N^{(i)} = N^{(j)}] \leq \varepsilon(l^{(i)}, l^{(j)}) + \frac{1}{2^n} \times b^{(j)}, \qquad (77)$$

by the same argument.

From (74), (75), (76) and (77), $\Pr[N^{(i)} = N^{(j)}] \leq \varepsilon(l^{(i)}, l^{(j)}) + \frac{1}{2^n} \times b^{(i)} \times b^{(j)}$.
Hence,

$$\begin{aligned}\Pr[\mathsf{Bad}_1] &\leq \textstyle\sum_{1 \leq i < j \leq q} \Pr[N^{(i)} = N^{(j)}]\\ &\leq \textstyle\sum_{1 \leq i < j \leq q} (\varepsilon(l^{(i)}, l^{(j)}) + \frac{1}{2^n} \times b^{(i)} \times b^{(j)})\\ &\leq \textstyle\sum_{1 \leq i < j \leq q} \varepsilon(l^{(i)}, l^{(j)}) + \frac{1}{2^n} \times \sum_{1 \leq i < j \leq q} (b^{(i)} \times b^{(j)}); \qquad (78)\end{aligned}$$

The property of the hash function that it is $(\varepsilon, \varepsilon')$-eligible, gives

$$\begin{aligned}\Pr[\mathsf{Bad}_2] &\leq \textstyle\sum_{1 \leq k \leq q} \Pr[N^{(k)} \in \mathsf{Dom}_2]\\ &\leq \min(q, \#\mathcal{L}) \times \textstyle\sum_{1 \leq k \leq q} \varepsilon'(l^{(k)}); \qquad (79)\end{aligned}$$

From the fact that $\mathsf{Q}^{(i)}$s are uniform random variables, we have

$$\begin{aligned}\Pr[\mathsf{Bad}_3] &\leq \textstyle\sum_{1 \leq k \leq q} \Pr[N^{(k)} \in \mathsf{Dom}_3]\\ &\leq \frac{q \times \mathsf{nBlks}}{2^n}. \qquad (80)\end{aligned}$$

So, from (78), (79) and (80), we get,

$$\begin{array}{rcl} \Pr[\mathsf{Bad}] & = & \Pr[\mathsf{Bad}_1 \vee \mathsf{Bad}_2 \vee \mathsf{Bad}_3] \\ & \leq & \sum_{1\leq i<j\leq q} \varepsilon(l^{(i)}, l^{(j)}) + \frac{1}{2^n} \times \sum_{1\leq i<j\leq q} (b^{(i)} \times b^{(j)}) \\ & + & \min(q, \#\mathcal{L}) \times \sum_{1\leq k\leq q} \varepsilon'(l^{(k)}) + \frac{q\times \mathsf{nBlks}}{2^n}. \end{array} \tag{81}$$

Again,

$$\Pr[\mathcal{B}^{\mathsf{rnd}} \Rightarrow 1] \;\leq\; \Pr[\mathcal{B}^{\mathsf{rnd}} \Rightarrow 1 \wedge \overline{\mathsf{Bad}}] + \Pr[\mathsf{Bad}]. \tag{82}$$

Seen in conjunction with $\overline{\mathsf{Bad}}$, the random variables in Dom_1 take distinct values and these value are also distinct from the values of the random variables in Dom_2 and Dom_3. As a result, the outputs of the random oracle on the inputs from Dom_1 are independent and uniform random n-bit strings. So, we have

$$\Pr[\mathcal{A}^{\mathsf{rnd}} \Rightarrow 1 \wedge \overline{\mathsf{Bad}}] \;=\; \Pr[\mathcal{B}^{\mathsf{rnd}} \Rightarrow 1 \wedge \overline{\mathsf{Bad}}]. \tag{83}$$

Hence, as in the expression (60) obtained in the proof of Theorem 7, here we get,

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{E\text{-}PRF}}(\mathcal{A}) \;\leq\; \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{E}}(\mathcal{B}) + \Pr[\mathsf{Bad}]. \tag{84}$$

Using the upper bound on $\Pr[\mathsf{Bad}]$ from (81), we get,

$$\begin{array}{rcl} \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{E\text{-}PRF}}(\mathcal{A}) & \leq & \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{E}}(\mathcal{B}) + \displaystyle\sum_{1\leq i<j\leq q} \varepsilon(l^{(i)}, l^{(j)}) + \frac{1}{2^n} \times \sum_{1\leq i<j\leq q} (b^{(i)} \times b^{(j)}) \\ & & + \min(q, \#\mathcal{L}) \times \displaystyle\sum_{1\leq k\leq q} \varepsilon'(l^{(k)}) + \frac{q \times \mathsf{nBlks}}{2^n}. \end{array} \tag{85}$$

The resource bounded advantage follows from this on noting that the number of queries made by $\mathcal{B}$ is at most $2q + \mathsf{nBlks}$ and for each query made by $\mathcal{A}$, $\mathcal{B}$ has to calculate the function $\mathfrak{b}(\cdot)$, hash the message and do some basic computation and bookkeeping. □