Designs, Codes and Cryptography (2021) 89:709–736
https://doi.org/10.1007/s10623-020-00840-w

Variants of Wegman-Carter message authentication code supporting variable tag lengths

Sebati Ghosh[1] · Palash Sarkar[1]

Received: 27 May 2020 / Revised: 24 December 2020 / Accepted: 26 December 2020 /
Published online: 4 February 2021

Abstract
In this work, we study 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 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.

Keywords MAC · Variable tag length · Wegman-Carter · Security bound

Mathematics Subject Classification 94A60

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 [26] discussion pertaining to different tag lengths suggested for the MAC

Communicated by F. Mendel.

✉ Palash Sarkar
palash@isical.ac.in

Sebati Ghosh
sebati_r@isical.ac.in

1 Indian Statistical Institute, 203, B.T.Road, Kolkata 700108, India

scheme UMAC [15]. This scheme had the possibility of using 32-bit, 64-bit, 96-bit and 128-bit tags. Finney [12], 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 [13] explains the issue further and suggests how a valid 128-bit tag can be obtained with only about 2^{34} queries. Wagner [27] 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 [15] states that a "UMAC key (or session) must have an associated and immutable tag length". Another suggestion put forward by Finney [13] 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 [18] 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 [23] 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 Vizár [22].

Two concrete motivations are provided in [22] 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") [21] 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 consider the problem of obtaining secure variable tag length MAC schemes. The Wegman-Carter [29] 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. These attacks do not repeat nonces for tag generation queries. Among the attacked schemes is the scheme obtained by nonce stealing following the suggestion of Finney [12] 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.

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.

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 [29] 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 [25] and Bernstein [5]. Recently, the optimality of Bernstein's bound was established in [17,20].

The point that tag lengths can vary depending on the application has been noted in [24] where the problem of determining an economically optimal tag length has been considered from a game theoretic point of view. This is completely different from the work reported in the present paper.

Relation to the work of Reyhanitabar et al. [22]: The notion of authenticated encryption with associated data (AEAD) which can support variable tag lengths was introduced in [22]. An AEAD scheme has two algorithms, namely encryption and decryption. The encryption algorithm takes as input a nonce, a plaintext, an associated data and a tag length and returns the corresponding ciphertext; while the decryption algorithm takes as input a nonce, a ciphertext, an associated data and a tag length and either returns $\perp$ indicating that the input is improper, or, returns the corresponding plaintext. Such an AEAD scheme can be considered to be a nonce-based MAC scheme where the plaintext is always fixed to the empty string and the message to be authenticated is provided as the associated data. With this modification, the formalisation of the authenticity of the AEAD scheme in [22] turns out to be the same as the formalisation of the variable tag length nonce-based MAC scheme introduced in this work. The difference between our formalisation and that of [22] is in the treatment of adversarial resources. We have considered the notion of query profile, while the usual notion of query complexity has been considered in [22]. In terms of construction, the contribution of [22] is different from ours. A variant of OCB [16] is considered in [22], while we describe variants of the Wegman-Carter scheme.

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 $0 \le i < 2^k - 1$, $\mathsf{bin}_k(i)$ 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 $(K \overset{\$}{\leftarrow} \mathcal{K} : \mathcal{A}^{F_K(\cdot)} \Rightarrow 1)$ 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.

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

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 nonce-based 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 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 $\mathsf{nvMAC.Gen}(K, N, x, \lambda)$ and $\mathsf{nvMAC.Verify}(K, N, x, \mathsf{tag}, \lambda)$, 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 $\mathsf{nvMAC.Gen}_K(N, x, \lambda)$ to denote $\mathsf{nvMAC.Gen}(K, N, x, \lambda)$, and similarly $\mathsf{nvMAC.Verify}_K(N, x, \mathsf{tag}, \lambda)$ to denote $\mathsf{nvMAC.Verify}(K, N, x, \mathsf{tag}, \lambda)$. The inputs and outputs of $\mathsf{nvMAC.Gen}_K(N, x, \lambda)$ and $\mathsf{nvMAC.Verify}_K(N, x, \mathsf{tag}, \lambda)$ are as follows.

- $\mathsf{nvMAC.Gen}_K(N, x, \lambda)$:
  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.
- $\mathsf{nvMAC.Verify}_K(N, x, \mathsf{tag}, \lambda)$:
  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.

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

Security: The security for a (nonce-based) 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. $\mathcal{A}$ 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 $1 \le j \le q_v$, $\mathsf{xxx}_v^{(j)}$ is either true or false. The query profile of $\mathcal{A}$ is the list

$$\begin{aligned}\mathfrak{C} = (&q_g, q_v, (\mathfrak{n}_g^{(1)}, \mathfrak{m}_g^{(1)}, \lambda_g^{(1)}), \ldots, (\mathfrak{n}_g^{(q_g)}, \mathfrak{m}_g^{(q_g)}, \lambda_g^{(q_g)}), (\mathfrak{n}_v^{(1)}, \mathfrak{m}_v^{(1)}, \lambda_v^{(1)}),\\ &\ldots, (\mathfrak{n}_v^{(q_v)}, \mathfrak{m}_v^{(q_v)}, \lambda_v^{(q_v)}))\end{aligned} \tag{2}$$

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

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 $1 \le i < j \le q_g$. 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. Usual nonce-respecting behaviour requires the nonces in the tag generation queries to be distinct. By relaxing this condition, we provide the adversary with more power. So, a scheme proved secure against the weaker form of nonce-respecting behaviour maintains security even if nonces are repeated in tag generation queries as long as the (nonce, tag-length) pairs are distinct. The second 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 $\left(N_g^{(i)}, x_g^{(i)}, \mathsf{tag}_g^{(i)}, \lambda_g^{(i)}\right)$ for any i in $\{1, \ldots, q_g\}$.

The adversary makes a number of verification queries. The tag lengths of these queries could be different. There is no restriction on the adversary to choose a target tag length before making the queries to its oracles. For any tag length λ, the adversary is successful if a verification query for this tag length returns true. So, for any value of the tag length, there is a corresponding event that the adversary is successful for a particular tag length. Formally, 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 $\mathsf{nvMAC.Verify}_K\left(N_v^{(j)}, x_v^{(j)}, \mathsf{tag}_v^{(j)}, \lambda_v^{(j)}\right)$ returns true. For each $\lambda \in \mathcal{L}$, the adversary's advantage in breaking the authenticity of nvMAC is defined to be $\Pr[\mathsf{succ}_{\mathcal{A}}(\lambda)]$. This is written as follows.

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

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

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

Remark The security model allows nonces to be repeated with different tag lengths. As explained above, this provides the adversary with more power. We further note that allowing nonces to be reused with different tag lengths permits generation of fewer nonces which may be of interest in some resource-constrained applications. At this point though, we are unable to provide a concrete example.

Security in terms of query complexity: 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, $\sigma_g = \sum_{1\le i\le q_g}(\mathsf{len}(N_g^{(i)})+\mathsf{len}(x_g^{(i)})+\mathsf{len}(\lambda_g^{(i)})) = \sum_{1\le i\le q_g}(\mathfrak{n}_g^{(i)}+\mathfrak{m}_g^{(i)}+\mathsf{len}(\lambda_g^{(i)}))$ 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 query complexity (σ_g,σ_v), $\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{nvMAC}}[\lambda](t,\sigma_g,\sigma_v)$ is the maximum of $\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{nvMAC}}[\lambda](\mathcal{A})$ taken over all adversaries $\mathcal{A}$ running in time t and having query complexity (σ_g,σ_v).

Given a query profile $\mathfrak{C}$ of any adversary $\mathcal{A}$ the corresponding query complexity (σ_g,σ_v) can be readily derived in the above manner. On the other hand, it is to be noted that, various query profiles can have the same query complexity. Hence, in the security definition above in terms of query complexity, when one maximises over query complexity, the value obtained is the maximum over all possible query profiles which have that same query complexity. This gives us the following proposition.

Proposition 1 *Let us fix a query complexity (σ_g,σ_v) and let $\c{C}_{(\sigma_g,\sigma_v)}$ be the set of all query profiles having query complexity (σ_g,σ_v), i.e.,*

$$\c{C}_{(\sigma_g,\sigma_v)} := \{\mathfrak{C} : \textit{the query complexity of } \mathfrak{C} \textit{ is } (\sigma_g,\sigma_v)\}.$$

Then,

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{nvMAC}}[\lambda](t,\sigma_g,\sigma_v) = \mathsf{max}_{\mathfrak{C}\in\c{C}_{(\sigma_g,\sigma_v)}}\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{nvMAC}}[\lambda](t,\mathfrak{C}). \tag{4}$$

Later we explain the rationale for considering query profiles.

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 $\mathfrak{C}$, $\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{nvMAC}}[\lambda](\mathfrak{C})$ is the maximum of $\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{nvMAC}}[\lambda](\mathcal{A})$ taken over all adversaries $\mathcal{A}$ having query profile $\mathfrak{C}$.

3 Towards building a variable tag length 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 [29] 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.,

$$\mathsf{WC\text{-}nvMAC} : (N, x) \xrightarrow{(K,\tau)} \mathsf{F}_K(N) \oplus \mathsf{Hash}_\tau(x). \tag{5}$$

Below we argue that several natural extensions of WC-nvMAC are not secure. We assume that binary representation of tag lengths fit within a byte. 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 2 *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 (5) is to truncate the output, i.e.,

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

The scheme trunc is not secure as can be seen from the following attacks. Note that in this case the message space is $\mathbb{F}_{2^n}$.

Attack 1 on trunc: Let x be a message and N be a nonce. The adversary makes a tag generation query (N, x, n) and gets in response t. Now the adversary makes a verification query $(N, x, \mathsf{msb}_{n-1}(t), n-1)$ and it is successful with probability 1. Thus the adversary makes a successful forgery with only one tag generation query.

Attack 2 on trunc: Another attack which repeats nonces in tag generation queries and reveals more information is the following. 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 2, 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 two tag generation and one verification queries.

The first attack shows that a simple truncation of the Wegman-Carter MAC scheme does not work while the second attack shows that by repeating nonces in tag generation queries the hash key can be obtained. One possibility of modifying trunc is to apply F_K a second time before applying truncation, i.e., the tag is obtained as $\mathsf{msb}_\lambda(\mathsf{F}_K(\mathsf{F}_K(N) \oplus \mathsf{Hash}_\tau(x)))$.

The resulting scheme is also not secure. The first simple attack on trunc also works for this modified scheme.

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.

$$\mathsf{nvMAC\text{-}t1}_{K,\tau} : (N, x, \lambda) \xrightarrow{(K,\tau)} \mathsf{msb}_\lambda(\mathsf{F}_K(\mathsf{bin}_8(\lambda)||N) \oplus \mathsf{Hash}_\tau(x)). \quad (6)$$

$$\mathsf{nvMAC\text{-}t2}_{K,\tau} : (N, x, \lambda) \xrightarrow{(K,\tau)} \mathsf{msb}_\lambda(\mathsf{F}_K(N) \oplus \mathsf{Hash}_\tau(\mathsf{bin}_8(\lambda)||x)). \quad (7)$$

$$\mathsf{nvMAC\text{-}t3}_{K,\tau} : (N, x, \lambda) \xrightarrow{(K,\tau)} \mathsf{msb}_\lambda(\mathsf{F}_K(\mathsf{bin}_8(\lambda)||N) \oplus \mathsf{Hash}_\tau(\mathsf{bin}_8(\lambda)||x)). \quad (8)$$

$$\mathsf{nvMAC\text{-}Generic}_{(K_\lambda,\tau_\lambda)_{\lambda\in\mathcal{L}}} : (N, x, \lambda) \xrightarrow{(K_\lambda,\tau_\lambda)} \mathsf{msb}_\lambda(\mathsf{F}_{K_\lambda}(N) \oplus \mathsf{Hash}_{\tau_\lambda}(x)). \quad (9)$$

$$\mathsf{nvMAC\text{-}t4}_{(K_\lambda,\tau)_{\lambda\in\mathcal{L}}} : (N, x, \lambda) \xrightarrow{(K_\lambda,\tau)} \mathsf{msb}_\lambda(\mathsf{F}_{K_\lambda}(N) \oplus \mathsf{Hash}_\tau(x)). \quad (10)$$

$$\mathsf{nvMAC\text{-}t5}_{(K_\lambda,\tau)_{\lambda\in\mathcal{L}}} : (N, x, \lambda) \xrightarrow{(K_\lambda,\tau)} \mathsf{msb}_\lambda(\mathsf{F}_{K_\lambda}(N) \oplus \mathsf{Hash}_\tau(\mathsf{bin}_8(\lambda)||x)). \quad (11)$$

$$\mathsf{nvMAC\text{-}t6}_{(K,\tau_\lambda)_{\lambda\in\mathcal{L}}} : (N, x, \lambda) \xrightarrow{(K,\tau_\lambda)} \mathsf{msb}_\lambda(\mathsf{F}_K(N) \oplus \mathsf{Hash}_{\tau_\lambda}(x)). \quad (12)$$

$$\mathsf{nvMAC}_{(K,\tau_\lambda)_{\lambda\in\mathcal{L}}} : (N, x, \lambda) \xrightarrow{(K,\tau_\lambda)} \mathsf{msb}_\lambda(\mathsf{F}_K(\mathsf{bin}_8(\lambda)||N) \oplus \mathsf{Hash}_{\tau_\lambda}(x)). \quad (13)$$

Dependencies of input and/or key on λ for the above schemes are summarised in Table 1.

Nonce stealing: Finney [12] 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 nvMAC-t1, nvMAC-t3 and nvMAC use nonce stealing. While nvMAC is secure (as proved later), schemes nvMAC-t1 and nvMAC-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 nvMAC-t1, nvMAC-Generic, nvMAC-t4 and nvMAC-t6 to be $\mathbb{F}_{2^n}$, and that for the schemes nvMAC-t2, nvMAC-t3 and nvMAC-t5 to be $\mathbb{F}_{2^{n-8}}$.

Algorithm 1 describes an attack on nvMAC-t1 which uses findTag as a subroutine. In the attack, the tag generation and verification oracles are denoted by $\mathcal{O}_g$ and $\mathcal{O}_v$ respectively.

Table 1 For the schemes in (6) to (13), 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	
nvMAC-t1	yes	no	no	no	no
nvMAC-t2	no	no	yes	no	no
nvMAC-t3	yes	no	yes	no	no
nvMAC-Generic	no	yes	no	yes	yes
nvMAC-t4	no	yes	no	no	no
nvMAC-t5	no	yes	yes	no	no
nvMAC-t6	no	no	no	yes	no
nvMAC	yes	no	no	yes	yes

On being supplied with input (N, x, λ), the function $\mathsf{findTag}(N, x, \lambda)$ finds tag such that $(N, x, \mathsf{tag}, \lambda)$ passes the test by the verification oracle. To do this, $\mathsf{findTag}$ repeatedly queries the verification oracle, until a suitable tag is obtained. The expected number of queries made by $\mathsf{findTag}(N, x, \lambda)$ is 2^λ. Algorithm 1 invokes $\mathsf{findTag}$ with values of the tag length which are less than the target tag length.

The intuition behind the attack in Algorithm 1 is the 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 less 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 3. This divide-and-conquer attack strategy of using shorter tag length to learn information, with low cost, which is useful for longer tag lengths has previously been used in the context of AE [11,22].

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

```
1: set $\lambda \leftarrow n$;
2: choose $\lambda_1 \in \mathcal{L}$, such that $\lambda_1 < \lambda$;
3: choose distinct $N_1, N_2 \in \mathcal{N}$ and distinct $x_1, x_2, x_3, x_4 \in \mathcal{M}$;
4: $\mathsf{tag}^{(1)} \leftarrow \mathcal{O}_g(N_1, x_1, \lambda_1)$;
5: $\mathsf{tag}^{(2)} \leftarrow \mathsf{findTag}(N_1, x_2, \lambda_1)$;
6: set $\mathcal{C} \leftarrow \{\}$;
7: do
8:     choose $c \leftarrow \{0,1\}^{n-\lambda_1} \setminus \mathcal{C}$;
9:     set $\mathcal{C} \leftarrow \mathcal{C} \cup \{c\}$;
10:    using Proposition 2 solve $\mathsf{Hash}_\tau(x_1) \oplus \mathsf{Hash}_\tau(x_2) = (\mathsf{tag}^{(1)} \oplus \mathsf{tag}^{(2)})||c$
11:        for $\tau$ and let the solution be $\tau_c$;
12:    set $x_c \leftarrow \mathsf{tag}^{(1)} \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_c}(x_1))$;
13:    $\mathcal{R}_v^{(3)} \leftarrow \mathcal{O}_v(N_1, x_3, x_c \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_c}(x_3)), \lambda_1)$;
14: while $\mathcal{R}_v^{(3)} = \mathsf{false}$;
15: $\mathsf{tag}^{(4)} \leftarrow \mathcal{O}_g(N_2, x_4, \lambda)$;
16: choose any $x \in \mathcal{M} \setminus \{x_4\}$;
17: return $(N_2, x, \mathsf{Hash}_{\tau_c}(x) \oplus \mathsf{Hash}_{\tau_c}(x_4) \oplus \mathsf{tag}^{(4)}, \lambda)$.
```

```
findTag($N, x, \lambda$)
1: set $\mathcal{D} \leftarrow \{\}$;
2: do
3:     choose $\mathsf{tag} \leftarrow \{0,1\}^\lambda \setminus \mathcal{D}$;
4:     set $\mathcal{D} \leftarrow \mathcal{D} \cup \mathsf{tag}$;
5:     $\mathcal{R}_v \leftarrow \mathcal{O}_v(N, x, \mathsf{tag}, \lambda)$;
6: while $\mathcal{R}_v = \mathsf{false}$
7: return tag.
```

Proposition 3 *The attack given in Algorithm 1 on the scheme* nvMAC-t1 *given in (6) produces a forgery for tag length λ which is correct with probability 1. It requires one tag generation query and at most $2^{\lambda_1} + 2^{n-\lambda_1}$ verification queries on tag length λ_1 and one tag generation query and one verification query on tag length λ.*

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 17 is accepted, i.e. the corresponding response from $\mathcal{O}_v$ is true.
From Step 4 we get,

$$\mathsf{msb}_{\lambda_1}(\mathsf{F}_K(\mathsf{bin}_8(\lambda_1)||N_1) \oplus \mathsf{Hash}_\tau(x_1)) = \mathsf{tag}^{(1)}. \tag{14}$$

The $\mathsf{tag}^{(2)}$ returned by Step 5 satisfies

$$\mathsf{msb}_{\lambda_1}(\mathsf{F}_K(\mathsf{bin}_8(\lambda_1)||N_1) \oplus \mathsf{Hash}_\tau(x_2)) = \mathsf{tag}^{(2)}. \tag{15}$$

So, from (14) and (15) we get,

$$\mathsf{msb}_{\lambda_1}(\mathsf{Hash}_\tau(x_1) \oplus \mathsf{Hash}_\tau(x_2)) = \mathsf{tag}^{(1)} \oplus \mathsf{tag}^{(2)}. \tag{16}$$

Here $\mathsf{tag}^{(1)} \oplus \mathsf{tag}^{(2)}$ is a λ_1-bit binary string. Following Proposition 2, for each choice of c in the do-while loop in Steps 7 to 14, the equation in Step 10 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 (16) suggest that there is a correct c, such that the equation in Step 10 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,

$$\begin{aligned}
&\mathsf{nvMAC\text{-}t1}(N_1, x_3, \lambda_1)\\
&\quad= \mathsf{msb}_{\lambda_1}(\mathsf{F}_K(\mathsf{bin}_8(\lambda_1)||N_1) \oplus \mathsf{Hash}_{\tau_c}(x_3))\\
&\quad= \mathsf{tag}^{(1)} \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_c}(x_1)) \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_c}(x_3)) && (17)\\
&\quad= x_c \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_c}(x_3)). && (18)
\end{aligned}$$

The expression in (17) comes from (14) and that in (18) comes from Step 12 in Algorithm 1. Hence, in this particular iteration of the do-while loop, $\mathcal{R}_v^{(3)} = \mathsf{true}$ and the loop terminates.

Since $\lambda = n$, from Step 15 we obtain $\mathsf{F}_K(\mathsf{bin}_8(\lambda)||N_2) = \mathsf{Hash}_{\tau_c}(x_4) \oplus \mathsf{tag}^{(4)}$. For the choice of x in Step 16, i.e., $x \in \mathcal{M} \setminus \{x_4\}$ we have

$$\begin{aligned}
\mathsf{nvMAC\text{-}t1}(N_2, x, \lambda) &= \mathsf{F}_K(\mathsf{bin}_8(\lambda)||N_2) \oplus \mathsf{Hash}_{\tau_c}(x)\\
&= \mathsf{Hash}_{\tau_c}(x_4) \oplus \mathsf{tag}^{(4)} \oplus \mathsf{Hash}_{\tau_c}(x), && (19)
\end{aligned}$$

which is returned as the tag for (N_2, 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 15. The subroutine $\mathsf{findTag}$ makes a maximum of 2^{λ_1} verification queries on tags of lengths λ_1. The do-while loop in Steps 7 to 14 iterates at most $2^{n-\lambda_1}$ times for different values of c making a maximum of $2^{n-\lambda_1}$ verification queries on tags of lengths λ_1. The forgery returned in Step 17 is a verification query on a tag of length λ. Hence, the attack requires 2 tag generation queries and at most $2^{\lambda_1} + 2^{n-\lambda_1} + 1$ verification queries including the forgery. □

Remarks 1. One may note that this work considers variable length tags. So, the adversary can make verification queries for a particular tag length and provide a forgery for another tag length. The attack given in Algorithm 1 on the scheme $\mathsf{nvMAC\text{-}t1}$, forges the scheme with an n-bit tag, i.e. the attack is for tag length n; whereas, as shown in Proposition 3, the attack requires 2 tag generation queries and $2^{\lambda_1} + 2^{n-\lambda_1} + 1$ verification queries including the forgery, where $\lambda_1 < \lambda$. Among these queries, 1 tag generation query and $2^{\lambda_1} + 2^{n-\lambda_1}$ verification queries are with tag length λ_1. For example, suppose $n = 128$, and let $\lambda_1 = 64$. So, the attack uses $2^{65} + 1 < 2^{128}$ verification queries and produces a forgery for tag length 128. This constitutes a valid attack for tag length 128.

2. The security model for variable length tag nonce-based MAC allows nonces in tag generation queries to be repeated as long as the tag lengths are distinct. The attack in Algorithm 1 does not repeat nonces in tag generation queries. So, the scheme nvMAC-t1 is insecure even under the restriction that nonces in tag generation queries are distinct.

Insecurities of the schemes nvMAC-t1 to nvMAC-t5 follow from applications of Algorithm 1.

Attack on nvMAC-t2: Algorithm 1 works with the only modification that the forgery is changed to $(N_2, x, \mathsf{Hash}_{\tau_c}(\mathsf{bin}_8(\lambda)||x_4) \oplus \mathsf{tag}^{(4)} \oplus \mathsf{Hash}_{\tau_c}(\mathsf{bin}_8(\lambda)||x), \lambda)$.
Attack on nvMAC-t3: Algorithm 1 works with the only modification that the forgery is changed to $(N_2, x, \mathsf{Hash}_{\tau_c}(\mathsf{bin}_8(\lambda)||x) \oplus \mathsf{Hash}_{\tau_c}(\mathsf{bin}_8(\lambda)||x_4) \oplus \mathsf{tag}^{(4)}, \lambda)$.
Attack on nvMAC-t4: Algorithm 1 works with the only modification that the forgery is changed to $(N_2, x, \mathsf{Hash}_{\tau_c}(x) \oplus \mathsf{Hash}_{\tau_c}(x_4) \oplus \mathsf{tag}^{(4)}, \lambda)$.
Attack on nvMAC-t5: Algorithm 1 works with the only modification that the forgery is changed to $(N_2, x, \mathsf{Hash}_{\tau_c}(\mathsf{bin}_8(\lambda)||x) \oplus \mathsf{Hash}_{\tau_c}(\mathsf{bin}_8(\lambda)||x_4) \oplus \mathsf{tag}^{(4)}, \lambda)$.

The insecurity of nvMAC-t6 is discussed in Appendix A.

The scheme nvMAC-Generic can be considered to be a collection of $\#\mathcal{L}$ independent WC-nvMAC 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 proof from [5] applies to the individual schemes where the values of λ are fixed. 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, nvMAC-Generic cannot be considered to be a practical solution to the problem of obtaining a variable tag length MAC scheme.

The first step towards reducing key size is taken in the scheme nvMAC which uses a single key K for F and independent keys τ_λ. In the next section, we prove nvMAC to be secure and also consider further variants with smaller keys.

Remark Suppose nvMAC-t1 is modified to obtain a scheme nvMAC-t1$'$ in the following manner. The tag is obtained as $\mathsf{msb}_\lambda(\mathsf{F}_K(\mathsf{F}_K(\mathsf{bin}_8(\lambda)||N) \oplus \mathsf{Hash}_\tau(x)))$, i.e., a second application of F_K is made before truncating. It is not difficult to show that the scheme mapping (N, x, λ), under the key (K, τ), to the quantity $\mathsf{F}_K(\mathsf{F}_K(\mathsf{bin}_8(\lambda)||N) \oplus \mathsf{Hash}_\tau(x))$ is a PRF. It can be argued that nvMAC-t1$'$ is a secure variable tag length MAC scheme. However, the security bound for nvMAC-t1$'$ will be in the order of $q^2\varepsilon$, where the total number of queries is q and the hash function is ε-AXU. This bound is higher than the bounds obtained for the schemes that we consider. Hence, we do not consider nvMAC-t1$'$. In the above discussion, we have considered modification of nvMAC-t1 to nvMAC-t1$'$. The same comments apply to similar modifications of the other insecure schemes, namely nvMAC-t2 to nvMAC-t6.

4 Secure and efficient MAC schemes with variable length tag

We start with the scheme nvMAC given in (13). 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 nvMAC is $\mathcal{N} = \{0,1\}^{n-8}$ and the message space is $\mathcal{M}$. Let $\mathcal{L} \subseteq \{1, \ldots, \min(256, n)\}$ be the allowed set of tag lengths. Note that tag length equal to zero is not allowed and there are 256 possible values of the tag length that are supported. If

Table 2 A secure and efficient nvMAC scheme from a random function

```
$\mathsf{nvMAC.Gen}_{(\tau_\lambda)_{\lambda\in\mathcal{L}}}(N, x, \lambda)$
  $Q = f(\mathsf{bin}_8(\lambda - 1)||N);$
  $R = Q \oplus \mathsf{Hash}_{\tau_\lambda}(x);$
  $\mathsf{tag} = \mathsf{msb}_\lambda(R);$
return tag.
```

λ is the tag length, then $\lambda - 1$ is at most 255 and consequently fits within a byte. So, instead of encoding λ, we encode $\lambda - 1$. This is a modification that we make to the scheme given in (13). Note that larger (or smaller) values of $\#\mathcal{L}$ can be considered by suitably adjusting the length of the nonces. From a practical point of view, however, it is difficult to think of any application which would require $\#\mathcal{L}$ to be more than 256.

The key space for nvMAC 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 $\mathsf{tag}' = \mathsf{nvMAC.Gen}_{(\tau_\lambda)_{\lambda\in\mathcal{L}}}(N, x, \lambda)$; 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 nvMAC schemes.

Theorem 1 *In the scheme* nvMAC *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* $Q^{(i)}_{v,\lambda} = (N^{(i)}_{v,\lambda}, x^{(i)}_{v,\lambda}, \mathsf{tag}^{(i)}_{v,\lambda}, \lambda)$ *be the* i*-th verification query with tag length* λ. *Let* $\ell^{(i)}_{v,\lambda} = \mathsf{len}(x^{(i)}_{v,\lambda})$. *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* δ_i *be such that the* i*-interpolation probability of* f *is at most* $\delta_i/(2^n)^i$. *Then*

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{nvMAC}}[\lambda_0](t, \mathfrak{C}) \leq \frac{1}{2^{\lambda_0}} \times \sum_{1\leq i\leq q_{v,\lambda_0}} \gamma_i \tag{20}$$

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 nvMAC. 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 $\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{nvMAC}}[\lambda_0](t, \mathfrak{C}) = \Pr[\mathsf{succ}(\mathcal{A}(\mathcal{T}), \lambda_0)]$.

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,

$$\begin{aligned}\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{nvMAC}}[\lambda_0](\mathcal{A}) = \Pr[\mathsf{succ}(\mathcal{A}(\mathcal{T}), \lambda_0)] = \Pr[\mathcal{E} \vee \mathcal{S}] &= \Pr[\mathcal{E} \vee (\overline{\mathcal{E}} \wedge \mathcal{S})] \\ &= \Pr[\mathcal{E}] + \Pr[\overline{\mathcal{E}} \wedge \mathcal{S}].\end{aligned}$$

Given the adversary $\mathcal{A}$ and the transcript $\mathcal{T}$, we define two adversaries $\mathcal{A}'$ and $\mathcal{A}''$ and correspondingly two transcripts $\mathcal{T}'$ and $\mathcal{T}''$ in the following manner.

- Adversary $\mathcal{A}'$ is the same as $\mathcal{A}$ up to and including the first verification query for tag length λ_0; the transcript $\mathcal{T}'$ is obtained from $\mathcal{T}$ by dropping from $\mathcal{T}$ all queries after the first verification query for tag length λ_0. So, $\Pr[\mathsf{succ}(\mathcal{A}'(\mathcal{T}'), \lambda_0)] = \Pr[\mathcal{E}]$.
- Adversary $\mathcal{A}''$ is the same as $\mathcal{A}$ except for the first verification query for tag length λ_0. $\mathcal{A}''$ does not issue the first verification query for tag length λ_0. The transcript $\mathcal{T}''$ is the same as that of $\mathcal{T}$ except that in $\mathcal{T}''$, the answer to the first verification query for tag length λ_0 is set to be false[1]. The event $\overline{\mathcal{E}} \wedge \mathcal{S}$ captures the following situation for $\mathcal{A}$ on the transcript $\mathcal{T}$: the response to the first verification query for tag length λ_0 is false and $\mathcal{A}$ is successful on some later verification query for tag length λ_0. Note that this situation is exactly the event that $\mathcal{A}''$ is successful for tag length λ_0 on transcript $\mathcal{T}''$. So, $\Pr[\mathsf{succ}(\mathcal{A}''(\mathcal{T}''), \lambda_0)] = \Pr[\overline{\mathcal{E}} \wedge \mathcal{S}]$.

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 (20), 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

[1] Bernstein's proof in [5] for nonce-based MAC considers simulation of the first forgery attempt with the simulator returning true if the provided tag is equal to the tag returned by a previous tag generation query on the same nonce and message, and false otherwise. In our case, since we are disallowing useless queries, there could not have been a previous tag generation query for the tag length λ_0 with the same nonce and message as that of the first verification query for tag length λ_0. So, in our case, such a simulator would always return.false.

(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 $\mathfrak{N} = \mathsf{bin}_8(\lambda_0 - 1)||N$, $Q = f(\mathfrak{N})$ 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 $\mathfrak{N}^{(i)} = \mathsf{bin}_8(\lambda^{(i)} - 1)||N^{(i)}$ 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

$$\Pr[\mathsf{succ}(\mathcal{A}(\mathcal{T}), \lambda_0)] \leq \gamma / 2^{\lambda_0}. \tag{21}$$

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

$$\Pr[\mathsf{succ}(\mathcal{A}(\mathcal{T}), \lambda_0)] = \Pr[\mathsf{msb}_{\lambda_0}(Q \oplus \mathsf{Hash}_{\tau_0}(x)) = \mathsf{tag}]. \tag{22}$$

We consider the probability on the right hand side of (22) 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 denote this functional dependence. We would like to emphasise that the adversary does not have access to $\mathbf{Q}$ and $\boldsymbol{\tau}$ and writing x and tag as functions of $\mathbf{Q}$ and $\boldsymbol{\tau}$ is only to help in the argument. 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,

$$\begin{aligned}
&\Pr[\mathsf{msb}_{\lambda_0}(Q \oplus \mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \boldsymbol{\tau}))) = \mathsf{tag}(\mathbf{Q}, \boldsymbol{\tau})] \\
&\quad = \Pr[\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{Q}, \boldsymbol{\tau}) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \boldsymbol{\tau})))] \\
&\quad = \sum_{\mathbf{a}, a} \Pr[\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{Q}, \boldsymbol{\tau}) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \boldsymbol{\tau}))) \wedge (\boldsymbol{\tau} = \mathbf{a}) \wedge (\tau_0 = a)] \\
&\quad = \sum_{\mathbf{a}, a} \Pr[\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{Q}, \mathbf{a}) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{a}(x(\mathbf{Q}, \mathbf{a}))) \wedge (\boldsymbol{\tau} = \mathbf{a}) \wedge (\tau_0 = a)] \\
&\quad = \sum_{\mathbf{a}, a} \Pr[\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{Q}, \mathbf{a}) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{a}(x(\mathbf{Q}, \mathbf{a})))] \Pr[(\boldsymbol{\tau} = \mathbf{a}) \wedge (\tau_0 = a)].
\end{aligned} \tag{23}$$

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

$$\Pr[\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{Q}, \mathbf{a}) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{a}(x(\mathbf{Q}, \mathbf{a})))]$$

$$
\begin{aligned}
&= \sum_{\mathbf{b}} \Pr[\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{Q}, \mathbf{a}) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_a(x(\mathbf{Q}, \mathbf{a}))) \wedge (\mathbf{Q} = \mathbf{b})] \\
&= \sum_{\mathbf{b}} \Pr[\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{b}, \mathbf{a}) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_a(x(\mathbf{b}, \mathbf{a}))) \wedge (\mathbf{Q} = \mathbf{b})] \\
&= \sum_{\mathbf{b}} \Pr[\mathsf{msb}_{\lambda_0}(Q) = b \wedge (\mathbf{Q} = \mathbf{b})] \\
&\qquad \big(\text{where } b = \mathsf{tag}(\mathbf{b}, \mathbf{a}) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_a(x(\mathbf{b}, \mathbf{a})))\big) \\
&= \sum_{\mathbf{b}} \Pr[\mathsf{msb}_{\lambda_0}(f(\mathfrak{N})) = b, f(\mathfrak{N}^{(1)}) = b_1, \ldots, f(\mathfrak{N}^{(q)}) = b_q] \\
&= \sum_{\mathbf{b}} \sum_{c} \Pr[f(\mathfrak{N}) = b||c, f(\mathfrak{N}^{(1)}) = b_1, \ldots, f(\mathfrak{N}^{(q)}) = b_q] \\
&\le \sum_{\mathbf{b}} 2^{n-\lambda_0} \delta_{q+1}/(2^n)^{q+1} \\
&= 2^{n-\lambda_0} \delta_{q+1}/2^n = \gamma/2^{\lambda_0}. \qquad (24)
\end{aligned}
$$

Combining (23) and (24), we have

$$
\begin{aligned}
&\Pr[\mathsf{msb}_{\lambda_0}(Q \oplus \mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \boldsymbol{\tau}))) = \mathsf{tag}(\mathbf{Q}, \boldsymbol{\tau})] \\
&\quad = \sum_{\mathbf{a}, a} \Pr[\mathsf{msb}_{\lambda_0}(Q) = \mathsf{tag}(\mathbf{Q}, \mathbf{a}) \oplus \mathsf{msb}_{\lambda_0}(\mathsf{Hash}_a(x(\mathbf{Q}, \mathbf{a})))] \Pr[(\boldsymbol{\tau} = \mathbf{a}) \wedge (\tau_0 = a)] \\
&\quad \le \gamma/2^{\lambda_0} \sum_{\mathbf{a}, a} \Pr[(\boldsymbol{\tau} = \mathbf{a}) \wedge (\tau_0 = a)] = \gamma/2^{\lambda_0}. \qquad (25)
\end{aligned}
$$

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 $\mathfrak{N}^{(\star)} = \mathsf{bin}_8(\lambda^{(\star)} - 1)||N^{(\star)}$, $Q^{(\star)} = f(\mathfrak{N}^{(\star)})$ 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 $\mathsf{tag} \equiv \mathsf{tag}(\mathbf{Q}, \boldsymbol{\tau}, \mathsf{tag}^{(\star)})$. As in the earlier argument, we highlight that the adversary does not have access to $\mathbf{Q}$ and $\boldsymbol{\tau}$ and writing x and tag as functions of $\mathbf{Q}$ and $\boldsymbol{\tau}$ (and also $\mathsf{tag}^{(\star)}$) is to help in the argument. 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

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

$$= \sum_{\mathbf{a}}\Big(\sum_{t}\Pr[(\mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{\tau_0}(x(\mathbf{Q},\mathbf{a},\mathsf{t}))\oplus\mathsf{Hash}_{\tau_0}(x^{(\star)}(\mathbf{Q},\mathbf{a})))=\mathsf{tag}(\mathbf{Q},\mathbf{a},\mathsf{t})\oplus\mathsf{t})$$
$$\wedge(\mathsf{msb}_{\lambda_0}(Q)=\mathsf{tag}(\mathbf{Q},\mathbf{a},\mathsf{t})\oplus\mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{\tau_0}(x(\mathbf{Q},\mathbf{a},\mathsf{t}))))]\Big)\times\Pr[\boldsymbol{\tau}=\mathbf{a}]. \qquad (26)$$

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

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

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

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

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

$$\sum_{\mathbf{b},a}\Pr[(\mathsf{msb}_{\lambda_0}(\mathsf{Hash}_a(x)\oplus\mathsf{Hash}_a(x^{(\star)}))=\mathsf{tag}\oplus\mathsf{t})\wedge(\tau_0=a)]$$
$$\times\left(\sum_{c_1}\Pr[(Q=b)\wedge(\mathbf{Q}=\mathbf{b})]\right)$$
$$\le\sum_{\mathbf{b},a}\Pr[(\mathsf{msb}_{\lambda_0}(\mathsf{Hash}_a(x)\oplus\mathsf{Hash}_a(x^{(\star)}))=\mathsf{tag}\oplus\mathsf{t})\wedge(\tau_0=a)]\times 2^{n-\lambda_0}\frac{\delta_q}{(2^n)^q}$$
$$=2^{n-\lambda_0}\frac{\delta_q}{(2^n)^q}\times\sum_{\mathbf{b},a}\Pr[(\mathsf{msb}_{\lambda_0}(\mathsf{Hash}_a(x)\oplus\mathsf{Hash}_a(x^{(\star)}))=\mathsf{tag}\oplus\mathsf{t})\wedge(\tau_0=a)]$$

$$\begin{aligned}
&= 2^{n-\lambda_0}\delta_q/(2^n)^q \times \sum_{\mathbf{b}} \Pr[\mathsf{msb}_{\lambda_0}(\mathsf{Hash}_{\tau_0}(x) \oplus \mathsf{Hash}_{\tau_0}(x^{(\star)})) = \mathsf{tag} \oplus \mathsf{t}] \\
&= 2^{n-\lambda_0}\delta_q/(2^n)^q \times \sum_{\mathbf{b}} \sum_{c_2} \Pr[\mathsf{Hash}_{\tau_0}(x) \oplus \mathsf{Hash}_{\tau_0}(x^{(\star)}) = (\mathsf{tag} \oplus \mathsf{t})||c_2] \\
&\leq 2^{n-\lambda_0}\delta_q/(2^n)^q \times \sum_{\mathbf{b}} 2^{n-\lambda_0}\varepsilon(\ell, \ell^{(\star)}) \\
&= 2^{n-\lambda_0}\delta_q/(2^n)^q \times (2^n)^{q-1} \times 2^{n-\lambda_0}\varepsilon(\ell, \ell^{(\star)}) \\
&= 2^{n-2\lambda_0}\delta_q\varepsilon(\ell, \ell^{(\star)}). \qquad (27)
\end{aligned}$$

Combining (26) and (27), we have,

$$\begin{aligned}
&\Pr[\mathsf{msb}_{\lambda_0}(Q \oplus \mathsf{Hash}_{\tau_0}(x(\mathbf{Q}, \boldsymbol{\tau}, \mathsf{tag}^{(\star)}))) = \mathsf{tag}(\mathbf{Q}, \boldsymbol{\tau}, \mathsf{tag}^{(\star)})] \\
&\quad \leq \sum_{\mathbf{a}} \Big(\sum_{t} 2^{n-2\lambda_0}\delta_q\varepsilon(\ell, \ell^{(\star)}) \Big) \times \Pr[\boldsymbol{\tau} = \mathbf{a}] \\
&\quad = \sum_{t} 2^{n-2\lambda_0}\delta_q\varepsilon(\ell, \ell^{(\star)}) \times \sum_{\mathbf{a}} \Pr[\boldsymbol{\tau} = \mathbf{a}] \\
&\quad = 2^{\lambda_0}2^{n-2\lambda_0}\delta_q\varepsilon(\ell, \ell^{(\star)}) \\
&\quad = 2^{n-\lambda_0}\varepsilon(\ell, \ell^{(\star)})\delta_q = \gamma/2^{\lambda_0}. \qquad (28)
\end{aligned}$$

This proves the second case.

□

Tightness of the security bound: The scheme nvMAC 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 [17,20]. A natural question is to consider whether the bound of Theorem 1 is also tight. We have considered this question for nvMAC. It does not seem possible to use the proof approach used in [17,20] to show the tightness of the bound in Theorem 1. In fact, the approach does not also seem to work for the generic scheme nvMAC-Generic.

The security bound of Theorem 1 in terms of query complexity: The statement of Theorem 1 and the security bound provided in it are in terms of query profile. If it is to be translated to terms of query complexity, the following point is to be noted. The hash function $\{\mathsf{Hash}_\tau\}_{\tau\in\Theta}$ may be such that, the differential probability of the hash function may depend on the lengths of the particular queries. For example, if $\{\mathsf{Hash}_\tau\}_{\tau\in\Theta}$ is a polynomial hash, the degree of the polynomial formed from the messages and hence the corresponding differential probability is a function of the lengths of the messages. The details of this variation in the query lengths are lost when we move from the notion of query profile to the notion of query complexity. As a result, the variability in the differential probability also cannot be captured when the security is considered in terms of query complexity. In this case, a uniformity is required in the probability and to attain that, the maximum of all the differential probabilities is considered. As a result, the security bound obtained in terms of query complexity is not precise and depending on the particular queries made by the adversary, it may be an over-estimation by a large margin. Hence, in the detailed security analysis we consider the notion of query profile and the security in terms of query complexity has been mentioned in respective corollaries.

The statement of Theorem 1 and the security bound provided in it look as follows in terms of query complexity.

Corollary 1 *In the scheme* nvMAC *defined in Table 2, suppose that the hash function* $\{\mathsf{Hash}_\tau\}_{\tau\in\Theta}$ *be such that for any distinct* $x, x' \in \mathcal{M}$ *and any* $y \in \{0,1\}^n$, $\Pr[\mathsf{Hash}_\tau(x) \oplus \mathsf{Hash}_\tau(x') = y]$ *is at most* $\varepsilon \geq 1/2^n$, *i.e.* ε *is the maximum of the differential probabilities for all combination of messages.*

For $\lambda \in \mathcal{L}$, *let* $q_{g,\lambda}$ *(resp.* $q_{v,\lambda}$*) be the number of tag generation (resp. verification) queries for* λ. *Let the total number of bits in the tag generation queries be* σ_g *and that in the verification queries be* σ_v. *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* δ_q *be such that the* q*-interpolation probability of* f *is at most* $\delta_q/(2^n)^q$ *and* $(q+1)$*-interpolation probability of* f *is at most* $(\delta_q\varepsilon)/(2^n)^q$. *Then*

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{nvMAC}}[\lambda_0](t, \sigma_g, \sigma_v) \leq 2^{n-\lambda_0} q_{v,\lambda_0}\delta_q\varepsilon. \tag{29}$$

Essentially, in this case the bound is similar to the bound given in the security proof of the Wegman-Carter nonce-based MAC scheme given in [5]. If in some case the actual queries are such that the corresponding differential probabilities are much less than the maximum value, then this bound becomes much higher than the actual advantage of the adversary, i.e. the bound becomes more loose. Let us consider a numerical example to illustrate this scenario.

In this example, we will consider Horner's rule based hash function and the underlying field to be $\mathbb{F}_{2^n}$. The differential probability of the Horner's rule based hash for two distinct messages of length ℓ and ℓ', where $\ell \geq \ell'$, is given by $\varepsilon(\ell, \ell') = \ell/2^n$. For ease of understanding, in this example let us consider $\delta_q = 1$, which is true for a uniform random function. Let $n = 128$, $\lambda_0 = 96$, $q_{v,\lambda_0} = 1$. Let us consider an (rather artificial) upper limit of 2^{20} n-bit blocks on the length of the message the adversary can query on. We consider some scenarios and the corresponding query profile based advantages.

- **Scenario 1:** For tag length λ_0, let the adversary make 1 tag generation query and 1 verification query, each on a message containing 512 blocks. The differential probability reflected in the bound (20) is $\varepsilon(512, 512) = 2^9/2^{128}$ and the corresponding bound becomes 2^{-87}.
- **Scenario 2:** For tag length λ_0, let the adversary make 1023 tag generation queries and 1 verification query, each on a message containing 1 block. Let one of the tag generation queries have the same nonce as the verification query. Then, the differential probability reflected in the bound (20) is $\varepsilon(1, 1) = 1/2^{128}$ and the corresponding bound becomes 2^{-96}.
- **Scenario 3:** For tag length λ_0, let the adversary make one tag generation query and one verification query on messages having 2^{20} blocks and the same nonce. Then, the differential probability reflected in the bound (20) is $\varepsilon(2^{20}, 2^{20}) = 2^{20}/2^{128}$ and the corresponding bound becomes 2^{-76}.

Let us now consider the query complexity based advantage for the above scenarios. Looking at the bound in (29), we have no clue about which value of the differential probability to be used here. The reason is, in this case, we only have the information regarding the total query complexity, but we do not know the length of each message. As a result, we are forced to use the maximum value of the differential probability which is obtained for 2^{20}-block messages resulting in the differential probability to be $2^{20}/2^{128}$. The corresponding bound given by (29) in all three scenarios becomes 2^{-76}. So, we see that even though the query complexities in Scenarios 1 and 2 is 1024 blocks and the query complexity in Scenario 3 is 2^{21} blocks, the query complexity based advantage in all three cases are the same. This

Table 3 A secure and efficient nvMAC scheme using a stream cipher supporting an initialisation vector

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

illustrates that compared to the query complexity based advantage, the query profile based advantage provides a more granular information about the advantage.

It is to be noted that, the bound given by Bernstein [5] in the security proof of the Wegman-Carter nonce-based MAC scheme is $q_{v,\lambda_0}\delta_q\varepsilon$. This bound also lacks the information of particular message lengths. Hence, the difficulty stated above in case of complexity based advantage is applicable for this bound as well.

We have highlighted the differences between query profile based and query complexity based advantages. Also, we have provided bounds for both kinds of advantages. Depending on the requirement, one may use the appropriate kind of advantage and the corresponding bound.

4.1 Reducing key size

In a practical instantiation of nvMAC, 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 nvMAC 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 nvMAC.

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 [4,6,19] while typical examples of hash function where the key depends upon the length of the message is either the multi-linear hash [14], or the pseudo-dot product [30], 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 nvMAC scheme built from SC as SC-nvMAC. The tag generation algorithm for the SC-nvMAC scheme is shown in Table 3. The verification algorithm $\mathsf{SC\text{-}nvMAC.Verify}_K(N, x, \mathsf{tag}, \lambda)$ works as follows: compute $\mathsf{tag}' = \mathsf{SC\text{-}nvMAC.Gen}_K(N, x, \lambda)$; return true if $\mathsf{tag} = \mathsf{tag}'$, else return false.

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

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

Theorem 2 *In SC-nvMAC 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 $Q^{(i)}_{v,\lambda} = (N^{(i)}_{v,\lambda}, x^{(i)}_{v,\lambda}, \mathsf{tag}^{(i)}_{v,\lambda}, \lambda)$ be the i-th verification query with tag length λ. Let $\ell^{(i)}_{v,\lambda} = \mathsf{len}(x^{(i)}_{v,\lambda})$. 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. Then

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{SC\text{-}nvMAC}}[\lambda_0](t, \mathfrak{C}) \leq \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(t + t', q_g + q_v, n(q_g + q_v)) + \frac{1}{2^{\lambda_0}} \times \sum_{1\leq i\leq q_{v,\lambda_0}} \gamma_i \quad (30)$$

where $\gamma_i = 2^n \varepsilon(\ell^{(i)}_{v,\lambda_0}, \ell^{(i^\star)}_{g,\lambda_0})$ 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 $\gamma_i = 1$. 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 (30). Let the scheme resulting from the replacement be denoted as ρ-nvMAC.

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-nvMAC and having query profile $\mathfrak{C}$. It is required to upper bound $\mathsf{Adv}^{\mathrm{auth}}_{\rho\text{-}\mathsf{nvMAC}}[\lambda_0](\mathcal{A})$.

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 $\mathfrak{N}^{(i)} = \mathsf{bin}_8(\lambda^{(i)} - 1)||N^{(i)}$, $(Q^{(i)}, \tau_i) = \rho(\mathfrak{N}^{(i)})$ (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 nvMAC, the hash key depends upon the tag length, whereas in SC-nvMAC, 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 (30). □

The following corollary provides the translation of Theorem 2 in terms of query complexity.

Corollary 2 *In* SC-nvMAC *defined in Table 3, suppose that the hash function* $\{\mathsf{Hash}_\tau\}_{\tau\in\Theta}$ *be such that for any distinct* $x, x' \in \mathcal{M}$ *and any* $y \in \{0,1\}^n$, $\Pr[\mathsf{Hash}_\tau(x) \oplus \mathsf{Hash}_\tau(x') = y]$ *is at most* $\varepsilon \geq 1/2^n$, *i.e.* ε *is the maximum of the differential probabilities for all combination of messages.*

For $\lambda \in \mathcal{L}$, *let* $q_{g,\lambda}$ *(resp.* $q_{v,\lambda}$*) be the number of tag generation (resp. verification) queries for* λ. *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.*

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*

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{SC\text{-}nvMAC}}[\lambda_0](t, \sigma_g, \sigma_v) \leq \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(t + t', q_g + q_v, n(q_g + q_v)) + 2^{n-\lambda_0} q_{v,\lambda_0}\varepsilon. \quad (31)$$

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.*

In the scheme SC-nvMAC, the pair (Q, τ) is derived by applying the stream cipher to $\mathsf{bin}_8(\lambda - 1)||N$. 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 $\{\mathsf{F}_\mathsf{K}\}_{K\in\mathcal{K}}$, where for each $K \in \mathcal{K}$, $\mathsf{F}_K : \{0,1\}^n \to \{0,1\}^n$. Similar to the case of SC-nvMAC, 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 nvMAC. 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 nvMAC scheme built from the PRF F is shown in Table 4 and is denoted as F-nvMAC.Gen. The verification algorithm F-nvMAC.Verify$(N, x, \mathsf{tag}, \lambda)$ works as follows. Given $(N, x, \mathsf{tag}, \lambda)$, compute $\mathsf{tag}' = \mathsf{F\text{-}nvMAC.Gen}_K(N, x, \lambda)$; if $\mathsf{tag} =$

Table 4 A secure and efficient nvMAC scheme using a short output length PRF

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

tag', return true, else return false. In Table 4, F is used in a counter type mode of operation which was proposed in [28].
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.

The security of $\mathsf{F\text{-}nvMAC}$ is given by the following result.

Theorem 3 *In* $\mathsf{F\text{-}nvMAC}$ *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* $Q^{(i)}_{v,\lambda} = (N^{(i)}_{v,\lambda}, x^{(i)}_{v,\lambda}, \mathsf{tag}^{(i)}_{v,\lambda}, \lambda)$ *be the* i*-th verification query with tag length* λ. *Let* $\ell^{(i)}_{v,\lambda} = \mathsf{len}(x^{(i)}_{v,\lambda})$. *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. Then*

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{F\text{-}nvMAC}}[\lambda_0](t, \mathfrak{C}) \leq \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(t + t', B_g + B_v, n(B_g + B_v)) + \frac{(B_g + B_v)^2}{2^n} + \frac{1}{2^{\lambda_0}} \times \sum_{1\leq i\leq q_{v,\lambda_0}} \gamma_i \tag{32}$$

where

- $\gamma_i = 2^n \varepsilon(\ell^{(i)}_{v,\lambda_0}, \ell^{(i^\star)}_{g,\lambda_0})$ *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* $\gamma_i = 1$;

- $b^{(i)}_{v,\lambda} = \mathfrak{b}(x^{(i)}_{v,\lambda})$, $B_v = \sum_{\lambda} \sum_{1 \le i \le q_{v,\lambda}} (b^{(i)}_{v,\lambda} + 2)$;
- $b^{(i)}_{g,\lambda} = \mathfrak{b}(x^{(i)}_{g,\lambda})$, $B_g = \sum_{\lambda} \sum_{1 \le i \le q_{g,\lambda}} (b^{(i)}_{g,\lambda} + 2)$.

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 $\mathsf{bin}_8(\lambda - 1)||N$. 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 $\#\mathcal{D} \le B_g + B_v$. 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 $\Pr[\mathsf{bad}] \le (B_g + B_v)^2/2^n$.

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

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

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. □

The following corollary provides the translation of Theorem 3 in terms of query complexity.

Corollary 3 *In* F-nvMAC *defined in Table 4, suppose that the hash function* $\{\mathsf{Hash}_\tau\}_{\tau \in \Theta}$ *be such that for any distinct $x, x' \in \mathcal{M}$ and any $y \in \{0, 1\}^n$,* $\Pr[\mathsf{Hash}_\tau(x) \oplus \mathsf{Hash}_\tau(x') = y]$ *is at most $\varepsilon \ge 1/2^n$, i.e. ε is the maximum of the differential probabilities for all combination of messages.*

For $\lambda \in \mathcal{L}$, let $q_{g,\lambda}$ (resp. $q_{v,\lambda}$) be the number of tag generation (resp. verification) queries for λ. 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.

Let λ be such that $q_{g,\lambda}, q_{v,\lambda} \ge 1$ and for $1 \le i \le q_{g,\lambda}$, let $Q^{(i)}_{g,\lambda} = (N^{(i)}_{g,\lambda}, x^{(i)}_{g,\lambda}, \lambda)$ be the i-th tag generation query with tag length λ; for $1 \le i \le q_{v,\lambda}$, let $Q^{(i)}_{v,\lambda} = (N^{(i)}_{v,\lambda}, x^{(i)}_{v,\lambda}, \mathsf{tag}^{(i)}_{v,\lambda}, \lambda)$ be the i-th verification query with tag length λ.

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

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{F\text{-}nvMAC}}[\lambda_0](t,\sigma_g,\sigma_v) \le \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(t+t', B_g+B_v, n(B_g+B_v)) + \frac{(B_g+B_v)^2}{2^n} + 2^{n-\lambda_0} \times q_{v,\lambda_0}\varepsilon, \tag{33}$$

where $b^{(i)}_{v,\lambda} = \mathfrak{b}(x^{(i)}_{v,\lambda})$, $b^{(i)}_{g,\lambda} = \mathfrak{b}(x^{(i)}_{g,\lambda})$, $B_v = \sum_{\lambda} \sum_{1\le i\le q_{v,\lambda}} (b^{(i)}_{v,\lambda}+2)$ *and* $B_g = \sum_{\lambda} \sum_{1\le i\le q_{g,\lambda}} (b^{(i)}_{g,\lambda}+2)$. *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.*

5 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.

Acknowledgements We are grateful to the reviewers for their detailed reading of the paper and for providing helpful comments.

A Attack on nvMAC-t6

The attack is described in Algorithm 2.

Proposition 4 *The attack given in Algorithm 2 on the scheme* nvMAC-t6 *produces a forgery for tag length* λ *which is correct with probability 1. It requires at most* $2^{\lambda_1+1}+2^{n-\lambda_1}$ *verification queries on tag length* λ_1 *and one tag generation query and at most* $2^{n-\lambda_1}$ *verification queries on tag length* λ.

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 17 to 24 such that $\mathcal{R}^{(5)}_v = \mathsf{true}$, i.e. there is a verification query in Step 23 which succeeds.

From Steps 4 and 5, we get that

$$\mathsf{msb}_{\lambda_1}(\mathsf{F}_K(N_1) \oplus \mathsf{Hash}_{\tau_{\lambda_1}}(x_1)) = \mathsf{tag}^{(1)}. \tag{34}$$

$$\mathsf{msb}_{\lambda_1}(\mathsf{F}_K(N_1) \oplus \mathsf{Hash}_{\tau_{\lambda_1}}(x_2)) = \mathsf{tag}^{(2)}. \tag{35}$$

So,

$$\mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_{\lambda_1}}(x_1) \oplus \mathsf{Hash}_{\tau_{\lambda_1}}(x_2)) = \mathsf{tag}^{(1)} \oplus \mathsf{tag}^{(2)}. \tag{36}$$

Here $\mathsf{tag}^{(1)} \oplus \mathsf{tag}^{(2)}$ is a λ_1-bit binary string.

Following Proposition 2, for each choice of c_1 in the do-while loop in Steps 17 to 24, the equation in Step 10 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 (36) suggest that there is a correct c_1, such that the equation in Step 10 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,

Algorithm 2 Attack on nvMAC-t6 for $\lambda = n$:

```
1: set $\lambda \leftarrow n$;
2: choose $\lambda_1 \in \mathcal{L}$, such that $\lambda_1 < \lambda$;
3: choose any $N_1 \in \mathcal{N}$ and distinct $x_1, x_2, x_3, x_4, x \in \mathcal{M}$;
4: $\mathsf{tag}^{(1)} \leftarrow \mathsf{findTag}(N_1, x_1, \lambda_1)$;
5: $\mathsf{tag}^{(2)} \leftarrow \mathsf{findTag}(N_1, x_2, \lambda_1)$;
6: set $\mathcal{C}_1 \leftarrow \{\}$;
7: do
8:     choose $c_1 \leftarrow \{0,1\}^{n-\lambda_1} \setminus \mathcal{C}_1$;
9:     set $\mathcal{C}_1 \leftarrow \mathcal{C}_1 \cup \{c_1\}$;
10:    using Proposition 2 solve $\mathsf{Hash}_{\tau_{\lambda_1}}(x_1) \oplus \mathsf{Hash}_{\tau_{\lambda_1}}(x_2) = (\mathsf{tag}^{(1)} \oplus \mathsf{tag}^{(2)})||c_1$
11:       for $\tau_{\lambda_1}$ and let the solution be $\tau_{c_1}$;
12:    set $x_{c_1} \leftarrow \mathsf{tag}^{(1)} \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_{c_1}}(x_1))$;
13:    $\mathcal{R}_v^{(3)} \leftarrow \mathcal{O}_v(N_1, x_3, x_{c_1} \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_{c_1}}(x_3)), \lambda_1)$;
14: while $\mathcal{R}_v^{(3)} = \mathsf{false}$;
15: $\mathsf{tag}^{(4)} \leftarrow \mathcal{O}_g(N_1, x_4, \lambda)$;
16: set $\mathcal{C}_2 \leftarrow \{\}$;
17: do
18:    choose $c_2 \leftarrow \{0,1\}^{n-\lambda_1} \setminus \mathcal{C}_2$;
19:    set $\mathcal{C}_2 \leftarrow \mathcal{C}_2 \cup \{c_2\}$;
20:    solve $\mathsf{Hash}_{\tau_\lambda}(x_4) = \mathsf{msb}_{\lambda_1}(\mathsf{tag}^{(4)}) \oplus x_{c_1}||c_2$
21:       for $\tau_\lambda$ and let the solution be $\tau_{c_2}$;
22:    set $x_{c_2} \leftarrow (\mathsf{msb}_{\lambda_1}(\mathsf{tag}^{(4)}) \oplus x_{c_1}||c_2) \oplus \mathsf{tag}^{(4)}$;
23:    $\mathcal{R}_v^{(5)} \leftarrow \mathcal{O}_v(N_1, x, x_{c_2} \oplus \mathsf{Hash}_{\tau_{c_2}}(x), \lambda)$;
24: while $\mathcal{R}_v^{(5)} = \mathsf{false}$.
```

$$\begin{aligned}
&\mathsf{nvMAC\text{-}t6}(N_1, x_3, \lambda_1)\\
&\quad = \mathsf{msb}_{\lambda_1}(\mathsf{F}_K(N_1) \oplus \mathsf{Hash}_{\tau_{c_1}}(x_3))\\
&\quad = \mathsf{tag}^{(1)} \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_{c_1}}(x_1)) \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_{c_1}}(x_3)) &(37)\\
&\quad = x_{c_1} \oplus \mathsf{msb}_{\lambda_1}(\mathsf{Hash}_{\tau_{c_1}}(x_3)). &(38)
\end{aligned}$$

The expression in (37) comes from (34) and that in (38) comes from Step 12 in Algorithm 2. Hence, in this particular iteration of the do-while loop, $\mathcal{R}_v^{(3)} = \mathsf{true}$ and the loop terminates.

Noting that $\lambda = n$, from Step 15, we get

$$\mathsf{F}_K(N_1) \oplus \mathsf{Hash}_{\tau_\lambda}(x_4) = \mathsf{tag}^{(4)} \Rightarrow \mathsf{Hash}_{\tau_\lambda}(x_4) = \mathsf{tag}^{(4)} \oplus \mathsf{F}_K(N_1). \tag{39}$$

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

$$\mathsf{Hash}_{\tau_\lambda}(x_4) = \mathsf{msb}_{\lambda_1}(\mathsf{tag}^{(4)} \oplus \mathsf{F}_K(N_1))||c_2 = (\mathsf{msb}_{\lambda_1}(\mathsf{tag}^{(4)}) \oplus x_{c_1})||c_2. \tag{40}$$

For the correct choice of c_2, the correct values of τ_{c_2} and x_{c_2} are obtained in Steps 21 and 22 respectively. For the correct c_2, from (39) and (40), we get,

$$\mathsf{F}_K(N_1) = \mathsf{Hash}_{\tau_\lambda}(x_4) \oplus \mathsf{tag}^{(4)} = ((\mathsf{msb}_{\lambda_1}(\mathsf{tag}^{(4)}) \oplus x_{c_1})||c_2) \oplus \mathsf{tag}^{(4)}, \tag{41}$$

which equals x_{c_2} according to Step 22 in Algorithm 2. Hence,

$$\mathsf{nvMAC\text{-}t6}(N_1, x, \lambda) = \mathsf{F}_K(N_1) \oplus \mathsf{Hash}_{\tau_{c_2}}(x) = x_{c_2} \oplus \mathsf{Hash}_{\tau_{c_2}}(x). \tag{42}$$

The last equality follows from (41). From (42), it is clear that for the iteration of the do-while loop in Steps 17 to 24, in which the correct c_2 is used, $\mathcal{R}_v^{(5)} = \mathsf{true}$ with probability 1, which proves the first part of the Lemma.

Steps 4 and 5 each require at most 2^{λ_1} verification queries for tag length λ_1. Step 13 requires at most $2^{n-\lambda_1}$ verification queries for tag length λ_1. A tag generation query for tag length λ is made in Step 15 and at most $2^{n-\lambda_1}$ verification queries are made for tag length λ in Step 23. This shows the complexity of the attack. □

Remarks 1. With $\lambda = n$ suppose $\lambda_1 = n/2$. Then the adversary makes a maximum of $3 \cdot 2^{n/2}$ verification queries for tag length $n/2$, one tag generation query and at most $2^{n/2}$ verification queries for tag length n. It produces a forgery for tag length n which is correct with probability 1. So, this is a valid forgery attack for tag length n.
2. Algorithm 2 makes a single tag generation query. Hence, the issue of repeating nonces in tag generation queries does not arise.

References

1. Aumasson J.-P., Bernstein D.J.: Siphash: A fast short-input PRF. In: Galbraith S.D., Nandi M. (eds.) 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. Berbain C., Gilbert H.: On the security of IV dependent stream ciphers. In: Biryukov, Alex (ed.) FSE, volume 4593 of Lecture Notes in Computer Science, pp. 254–273. Springer, (2007).
3. Bernstein D.J.: The Salsa20 family of stream ciphers. http://cr.yp.to/papers.html#salsafamily. Document ID: 31364286077dcdff8e4509f9ff3139ad. Date: 2007.12.25.
4. Bernstein D.J.: The poly1305-aes message-authentication code. In: Gilbert H., Handschuh H. (eds.) 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, pp. 32–49. Springer, (2005).
5. Bernstein D.J.: Stronger security bounds for Wegman-Carter-Shoup authenticators. In: Cramer R. (ed.) EUROCRYPT, volume 3494 of Lecture Notes in Computer Science, pp. 164–180. Springer, (2005).
6. Bernstein D.J.: Polynomial evaluation and message authentication (2007). http://cr.yp.to/papers.html#pema.
7. Bernstein D.J., Chou T.: Faster binary-field multiplication and faster binary-field macs. In: Joux, Antoine, Youssef, Amr M. (eds), 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, pp. 92–111. Springer, (2014).
8. Black J., Halevi S., Krawczyk H., Krovetz T., Rogaway P.: UMAC: Fast and secure message authentication. In: Wiener M.J. (ed.) 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. Chakraborty D., Ghosh S., Sarkar P.: A fast single-key two-level universal hash function. IACR Trans. Symmetric Cryptol. **2017**(1), 106–128 (2017).
11. Dobraunig C., Eichlseder M., Mendel F., Schläffer M.: Remark on variable tag lengths and OMD. https://groups.google.com/g/crypto-competitions/c/sekKDsIJvwU/m/5_V_TzZQaWYJ?pli=1. Accessed 15 Nov 2019 (2014).
12. Finney H.: CFRG discussion on UMAC. https://marc.info/?l=cfrg&m=143336318427069&w=2. Accessed 15 Nov 2019 (2005).
13. Finney H.: CFRG discussion on UMAC. https://marc.info/?l=cfrg&m=143336318527072&w=2. Accessed 15 Nov 2019 (2005).
14. Gilbert E.N., MacWilliams F., Jessie S., Neil J.A.: Codes which detect deception. Bell Syst. Tech. J. **53**, 405–424 (1974).
15. Krovetz T.: UMAC: Message authentication code using universal hashing. https://tools.ietf.org/html/draft-krovetz-umac-05.html. Accessed 15 Nov 2019, (2005).

16. Krovetz T., Rogaway P.: The software performance of authenticated-encryption modes. In: Joux, Antoine (ed.) FSE, volume 6733 of Lecture Notes in Computer Science, pages 306–327. Springer, (2011).
17. Luykx A., Preneel B.: Optimal forgeries against polynomial-based macs and GCM. In: Nielsen J.B., Rijmen V. (eds.) 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).
18. Manger J.H.: Attacker changing tag length in OCB. https://mailarchive.ietf.org/arch/msg/cfrg/gJtV9FCw92MguqqhxrSNUyIDZIw. Accessed 15 Nov 2019, (2013).
19. McGrew D.A., Viega J.: The security and performance of the Galois/Counter Mode (GCM) of operation. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT, volume 3348 of Lecture Notes in Computer Science, pages 343–355. Springer (2004).
20. Nandi M.: Bernstein bound on WCS is tight—repairing luykx-preneel optimal forgeries. In: Shacham H., Boldyreva A. (eds.) 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, pp. 213–238. Springer (2018).
21. Ounsworth M.: Footguns as an axis of security analysis. https://groups.google.com/a/list.nist.gov/forum/#!topic/pqc-forum/l2iYk-8sGnI. Accessed 15 Nov 2019, (2019).
22. Reyhanitabar R., Vaudenay S., Vizár D.: Authenticated encryption with variable stretch. In: Cheon J.H., Takagi T. (eds.) 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, pp. 396–425 (2016).
23. Rogaway P., Wagner D.: A critique of ccm. Cryptology ePrint Archive, Report 2003/070, (2003). https://eprint.iacr.org/2003/070.
24. Safavi-Naini R., Lisý V., Desmedt Y.: Economically optimal variable tag length message authentication. In: Kiayias A. (ed.) Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3–7, 2017, Revised Selected Papers, volume 10322 of Lecture Notes in Computer Science, pp. 204–223. Springer (2017).
25. Shoup V.: On fast and provably secure message authentication based on universal hashing. In: Koblitz, Neal (ed.) CRYPTO, volume 1109 of Lecture Notes in Computer Science, pages 313–328. Springer (1996).
26. UMAC. CFRG discussion on UMAC. http://marc.info/?l=cfrg&m=143336318427068&w=2. Accessed 15 Nov 2019, (2005).
27. Wagner D.: CFRG discussion on UMAC. https://marc.info/?l=cfrg&m=143336318527073&w=2. Accessed 15 Nov 2019, (2005).
28. Wang P., Feng D., Wenling W.: HCTR: A variable-input-length enciphering mode. In: Feng D., Lin D., Yung M. (eds.) CISC, volume 3822 of Lecture Notes in Computer Science, pp. 175–188. Springer, (2005).
29. Wegman Mark N.: Carter, Larry: New hash functions and their use in authentication and set equality. J. Comput. Syst. Sci. **22**(3), 265–279 (1981).
30. Winograd S.: A new algorithm for inner product. IEEE Trans. Comput. **17**, 693–694 (1968).