On Random Read Access in OCB

Ashwin Jha, Cuauhtemoc Mancillas-López, Mridul Nandi, and Sourav Sen Gupta

***Abstract*— Offset codebook or OCB mode is a popular block cipher mode of operation for authenticated encryption. The latest version of this cipher, called OCB3, is one of the finalists in CAESAR. In this paper, we explore the scope of random read access and out-of-sequence decryption in OCB. We observe that the current versions of OCB are inefficient in this respect owing to the ineptness of the underlying mask generating function (MGF). We propose new candidates for MGF based on AES round function, which are efficient in direct computation and provide comparable performance in the usual setting. Our schemes are not the obvious choices for MGF in conventional sense as they do not have optimal almost XOR universal (AXU) bound. In existing OCB designs, the MGFs are required to have 2^{-n}, i.e. optimal, AXU bound in order to upper bound the distinguishing advantage to $O(\sigma^2/2^n)$, where n is the block size of the underlying block cipher and σ is the total number of blocks among all queries. We find this specific requirement too restrictive. We abstract the OCB design, termed as GOCB, to look into the universal notion required from the underlying MGF. We propose a relaxed notion of AXU, called locally imperfect XOR universal (LIXU) hash, which can be of independent interest. Using LIXU as the underlying MGF, we recover reasonable security bounds for our schemes.**

***Index Terms*— AES, AXU, masking, OCB, universal hash.**

I. Introduction

AUTHENTICATED encryption or AE schemes [1]–[3] are symmetric-key algorithms which can simultaneously achieve data confidentiality, integrity and authentication. In recent years AE schemes have received considerable research interest, owing to the ongoing CAESAR competition [4] which aims to deliver a portfolio of state-of-the-art authenticated encryption schemes. Even though there are multiple designs based on pseudo-random generators [5], [6] and sponge-like functions [7], majority of the AE schemes have an underlying block cipher as their core component, and they rely on a secure and efficient mode of operation to achieve domain extension. In fact, 3 out of the 7 finalists in the CAESAR competition are based on an underlying (tweakable) block cipher. Some of the known block cipher based AE schemes are OCB [3], [8], [9], GCM [10], COLM [11]–[13] and Deoxys [14], [15].

OFFSET CODEBOOK or OCB mode [3], [8], [9] by Rogaway et al. is a single-pass and parallelizable block cipher mode of operation for nonce-based authenticated encryption (with associated data). OCB achieves both authentication and encryption with minimal performance overhead compared to classical single-pass encryption only modes such as CTR mode [16]. OCB requires one block cipher call per block of message, apart from a few constant number of calls for auxiliary tasks. In contrast, CCM [17] requires two block cipher calls per block of message, and GCM [10] requires one block cipher call and one field multiplication per block of message. As a result, OCB is approximately twice as fast [9], [18], [19] as any CTR mode based AE schemes, particularly CCM and GCM. In fact OCB3 [9] has, to a larger extent, already achieved the theoretical bounds for efficiency in software implementation.

A. Random Read Access and Out-of-Sequence Decryption

A block cipher based encryption mode of operation is said to have random read access feature, if it allows efficient decryption of any arbitrary encrypted block of message. This property could be beneficial in secure storage of read only data on cloud servers. The encrypted storage helps in avoiding unauthorized access, and the authentication tag helps in maintaining the integrity of the stored data. For example, say some enterprise application stores day to day system log files on some cloud server. Here the day timestamp may act as the nonce value. Now suppose on a later day the application faces some issue which warrants some portion of an encrypted log file. In this case the system administrator should be able to repeatedly read arbitrary blocks from the decrypted log, with as little overhead as possible, by simply mentioning the relevant day timestamp and the offset in the log file.

Yet another use-case for random read access is in disk encryption where access to an arbitrary block of some disk sector is highly beneficial. A popular and standardized disk encryption scheme, called XTS by Rogaway [8], is quite similar to OCB encryption.

A related feature is *out-of-sequence decryption where the mode of operation allows for the decryption of the encrypted block data stream in an arbitrary order*. In other words, the original ordering of the ciphertext blocks is not necessary for the decryption phase. If needed, the decrypted ciphertext can be reordered to get the original plaintext. This is quite useful in secure data transmission over connection-less network protocols such as User Datagram Protocol (UDP) [20], which allows for out-of-order delivery of data packets.

Manuscript received August 29, 2018; revised April 18, 2019; accepted June 12, 2019. Date of publication June 27, 2019; date of current version November 20, 2019.

A. Jha and M. Nandi are with the Applied Statistics Unit, Indian Statistical Institute, Kolkata 700108, India (e-mail: ashwin.jha1991@gmail.com; mridul.nandi@gmail.com).

C. Mancillas-López is with the Department of Computer Science, CINVESTAV-IPN, México City 07360, Mexico (e-mail: cuauhtemoc.mancillas@cinvestav.mx).

S. Sen Gupta is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798 (e-mail: sg.sourav@ntu.edu.sg).

Communicated by A. Canteaut, Associate Editor for Cryptography.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2019.2925613

If the underlying encryption scheme offers efficient out-of-sequence decryption then the data packets can be decrypted in on-the-fly manner; otherwise the received packets have to be reordered first which adds to the network protocol overhead. Yet another application area is real time data streaming protocols such as Secure Real-time Transport Protocol (SRTP) [21]–[23] and SRTP Control Protocol (SRTCP) [24], where on-the-fly decryption of received data packets is necessary to maintain the continuity of audio or visual data. In general, a scheme with random read access feature also allows out-of-sequence decryption.

Both CCM and GCM, being CTR based designs, allow for random read access: given the nonce and the block index, a single inverse block cipher call is required to access the message block. In case of OCB: given the nonce-based mask and the block index, a single block cipher call and a single MGF call is required to access the message block. So if the MGF is efficient in direct computation, then OCB can achieve random read access almost as efficiently as CCM and GCM.

We remark that random read access and out-of-sequence decryption are some features, which are defined exclusively for encryption schemes, and as such they allow release of unverified plaintext. It is well-known that OCB, CCM and GCM are trivially broken, when the adversary gets access to unverified plaintext. Still, we believe that there are practical use-cases, where random read access makes sense in context of authenticated encryption schemes as well. For instance, consider the use-case pertaining to encrypted log files on a cloud server. Here the cloud server may play the role of an adversary, in which case, it is clear that the adversary does not get access to unverified plaintext. Further, usually the logs follow some specific format for recording various events. If the encryption algorithm is secure, then it is almost impossible for the adversary to guess the right format. If the decryption algorithm receives unformatted log after decrypting few blocks, it can be sure that the cloud has tempered with the stored data.

B. Motivation

In connection with our discussion thus far, we observe a serious drawback in the instantiation of MGF in existing OCB designs. In all the existing OCB designs, the MGFs are instantiated with small-domain AXU hash functions. These functions are indeed quite efficient when computed sequentially over the indices; but they are not that efficient for direct computation at any arbitrary index, which renders the resulting OCB scheme inefficient in random read access. Take the use of xtimes in [3], [8], [9], [25] for example – the α-multiplication[1] is quite efficient in sequential computation as it requires only one shift and one conditional bit-wise XOR; but direct computation of xtimes for an arbitrary index i is not efficient at all, as it requires $O(\log i)$ many field multiplications if executed naively, or at least one field-multiplication even if all α^i values are precomputed, which is only possible for moderate values of i. Similar degradation is observed for other small-domain AXU hash functions such as Gray-code based hash [9] and Matrix-powered hash [26], [27]. In general, for most of the standard small-domain AXU hash functions: sequential computation requires just a few shift and XOR operations, whereas direct computation requires field multiplications or some other costly operations. This raises a simple question:

> (*a*) *Can we have efficient directly computable small-domain hash functions?*

A possible approach is to use the differential properties of data-dependent rotation (DDR) [9], [28], [29]. In fact OCB3 employs such a hash function for a very small domain of size 64, called stretch-then-shift [9], which requires four shifts and two XORs. Although DDR involves simple bit manipulation operations, the number of such operations are considerably higher to get a good AXU bound over a sufficiently large domain. For example the circulant hash [29] by Minematsu requires approx. 64 rotations and 64 XORs to get 2^{-128} AXU bound over $\mathbb{F}_{2^{64}}$. Furthermore, DDR is marred by patents, high cost in hardware and difficulty to get constant-time implementation.

Yet another approach is to use iterations of some cryptographic round function with good differential probability. Particularly, 4 rounds of AES offers very low differential probability bound [30]–[32] and most of the modern processors provide in-built support [33] for AES round function. This has lead to some efficient AES-based A(X)U hash functions [34]–[36]. While the use of reduced-round block ciphers could be an elegant solution, the number of rounds to get sufficient AXU bound is still a problem. For instance, on Intel Skylake processors, field multiplication is almost as good as four rounds of AES, which defeats the whole purpose of using AES rounds for MGF. For lesser number of rounds the performance is much better but the AXU bound is, in general, significantly low.

In view of this shortcoming of existing optimal AXU hash functions as MGFs, we rethink the basic philosophy behind masking in OCB like designs, and question the requirement of optimum AXU bounds in constructing the MGF. In existing proof techniques [8], [9], [37] the distinguishing advantage is upper bounded by $O(\sigma^2\epsilon)$, where σ is the total number of blocks in all queries and ϵ is the AXU bound of the MGF. Clearly, a sub-optimal ϵ will result in a sub-optimal bound for OCB. The immediate question that comes to mind is as follows:

> (*b*) *Can we relax the AXU condition on underlying MGFs to allow option for efficient direct computation while maintaining comparable security?*

In other words, we ask if one may reduce the AXU security margin of an underlying MGF without degrading the actual security of the OCB mode. More fundamentally, we try to investigate and establish the *practical requirements from a hash function* in terms of masking, rather than just accepting a standard off-the-shelf hash function with an optimum AXU bound from the literature.

[1] Here α denotes the primitive element of the underlying field.

C. Contributions

We answer the aforesaid questions (a) and (b) in affirmative, and hence our contributions are twofold:

- First, we propose two new 128-bit hash functions, called aes1 and aes2, based on 1 and 2 rounds of AES. We show that aes1 is a 2^{-96}-AXU hash over $\mathbb{F}_{2^8}$, and aes2 is a 2^{-113}-AXU hash over $\mathbb{F}_{2^{32}}$. On Intel Skylake processor: in direct computation, aes1 and aes2 are expected to be about 5 and 2.5 times faster than field multiplication. Similar improvements are expected for gray code and matrix based hash functions. In sequential computation aes1 and aes2 are slower than gray, but the degradation can be significantly reduced on modern processors where AES pipelining is possible.
  Note that if we use previous proof techniques, we get bounds of $O(\sigma^2/2^{96})$ and $O(\sigma^2/2^{113})$ for OCB with aes1 and aes2, which are significantly worse then OCB3 bound, $O(\sigma^2/2^{128})$.
- Second, we introduce a generic view of OCB (termed GOCB), using the modular notion of Mask Generating Functions (MGFs), to study the alternatives to AXU hash functions for masking. We introduce a different notion of universal hash in the context of mask generation — the notion of *Locally-Imperfect XOR Universal Hash* (LIXU) — and prove that this notion is sufficient to provide adequate (comparable to existing) security guarantees for GOCB. The basic idea is to model a mask generating hash function H whose differential probability depends on grouping (quantified by the chunk number) of the given inputs. More precisely, for an (ϵ, r)-LIXU hash H the domain can be partitioned into several disjoint sets of size at most r, such that H has optimum differential probability when the inputs x and y belong to distinct partitions and the probability degrades to at most $\epsilon \geq 0$ when x and y belong to the same partition.
  With respect to OCB, the standard notion of AXU uses a one-dimensional view of the message blocks, where the goal is to bound the differential probability between the masks of any two blocks among all σ many blocks. On the contrary, the notion of LIXU adopted by us views the message blocks conveniently as a two-dimensional array, where the differential probability is different within a row and across the rows. This view allows us to control the differential probability within a row by reducing the number of blocks (corresponds to the partition size r) in each message, while the differential probability across rows is controlled by the use of an encrypted nonce, as is the case with a regular AXU hash function. In fact, the number of *local* pairs of message blocks within a row may be controlled to be much less ($\leq r$) compared to the overall number of message blocks, such that even a low ϵ is adequately balanced in the effective security margin. This is the fundamental difference between an AXU hash and a LIXU hash, resulting in the benefits of the latter in terms of efficiency.
  We obtain following security bound for GOCB, in Theorem 1, where the underlying MGF is an (ϵ, r)-LIXU hash function:
  $$\mathbf{Adv}^{\mathsf{naead}}_{\mathsf{GOCB}[\Pi,\lambda]} \leq \frac{\sigma^2 + 3q\sigma + 2\sigma\widetilde{\sigma} + 2\widetilde{q}}{2^n} + (\sigma + 4\widetilde{\sigma})r\epsilon.$$
  Based on this generalization, we recover the security bound of $O(\sigma^2 2^{-128})$ for GOCB based on aes1 and aes2 when the maximum message length is restricted to 2^6 and 2^{30} blocks, respectively. Note that, we take $r = 1$ when $\epsilon = \Theta(2^{-n})$, as this corresponds to a perfect AXU hash function.

STRUCTURE OF THE PAPER — In Section II, we define the preliminary notions and notation to aid the technical results, and Section III documents the existing constructions of small-domain AXU hash functions in the literature. In Section IV we propose our AES based hash functions and derive concrete AXU bounds for them. In Section V, we introduce the generic view of OCB, as well as the technical notion of LIXU hash functions. Section V also contains the main theorem on the security bound of GOCB using a LIXU hash function as MGF, and the concrete bounds for GOCB instantiation using aes1 and aes2. The security proofs are delegated to Section VI. Section VII gives a comparison between the performance of OCB3, and GOCB instantiated with aes1 and aes2 with respect to sequential and random access implementations. Finally, Section VIII concludes the paper.

II. PRELIMINARIES

NOTATION — In this paper, we abide by the following set of notation:

- $\mathbb{N}$ denotes the set of non-negative integers and $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$ the positives.
- For $n \in \mathbb{N}^+$, $[n]$ denotes the set $\{1, \ldots, n\}$.
- For $m, n \in \mathbb{N}$ and $m \leq n$, $[m..n]$ denotes the set $\{m, \ldots, n\}$.
- For $m \in \mathbb{N}^+$, I_m denotes the identity matrix of size m.
- For any set $\mathcal{S}$ and $m \in \mathbb{N}$, $\mathcal{S}^m$ denotes the set of all m-tuples from $\mathcal{S}$, and $\mathcal{S}^{\underline{m}}$ denotes the set of all m-tuples with distinct elements from $\mathcal{S}$.
- We let $\mathcal{S}^{\leq \ell} = \cup_{i=1}^{\ell} \mathcal{S}^i$, $\mathcal{S}^+ = \cup_{i=1}^{\infty} \mathcal{S}^i$, and $\mathcal{S}^* = \cup_{i=0}^{\infty} \mathcal{S}^i$.

Throughout this paper, we fix $n \in \mathbb{N}^+$ as the block size, and the elements of $\{0,1\}$ and $\{0,1\}^n$ (often denoted by $\mathcal{B}$) are called *bits* and *blocks*, respectively. We use $\oplus$ and $\odot$ to denote the field addition (XOR) and field multiplication, respectively, over the finite field $\mathbb{F}_{2^n} = \mathcal{B} = \{0,1\}^n$. Throughout the paper, α denotes a primitive element of $\mathbb{F}_{2^n}$, and for any $x \in \mathbb{F}_{2^n}$, the operation $x \odot \alpha$ will be referred to as the α-multiplication operation on x.

REPRESENTATION OF STRINGS — If x is a vector (or a sequence or string) over a finite set $\mathcal{S}$ (coordinates are from $\mathcal{S}$), then $|x|$ typically denotes its length. For $i \in [|x|]$, x_i denotes its i-th coordinate.

- For $i, j \in [|x|]$ and $i \leq j$, we use the shorthand $x_{i..j} = (x_i, \ldots, x_j)$.
- For two strings x and y, $x||y$ denotes the concatenation of x and y.
- If $z = x\|y$, then x and y are the *prefix* and *suffix*, respectively of z.

- Strings over $\{0,1\}$ and $\{0,1\}^n$ are called binary and block strings respectively, and we will only consider binary and block strings in this paper.
- For a string b and $m \in \mathbb{N}$, b^m denotes the m-wise concatenation of b, i.e $b\|b\|\cdots$ (m times), this being the null string when $m = 0$.
- For some $m \in \mathbb{N}^+$, for $x \in \{0,1\}^m$, $b \in [m]$, $x \gg b$ denotes x shifted b bits to the right, i.e., $0^b\|x_{1..m-b}$, while $x \ll b$ denotes x shifted b bits to the left.

NOMENCLATURE FOR BLOCKS — For a binary string x, if $|x| = n$, we call x a complete block; if $|x| < n$, we call x an incomplete block; if $|x| = 0$, we call x the null string or the empty block. By convention, the empty block will also be referred as an incomplete block. Any binary sequence x can be mapped uniquely to a block sequence $\hat{x} = (\hat{x}_1, \ldots, \hat{x}_\ell, \hat{x}_*) \xleftarrow{n} x$, where $\ell \in \mathbb{N}$, $\hat{x}_1, \ldots, \hat{x}_\ell$ are complete blocks, and $\hat{x}_*$ is an incomplete (possibly empty) block, such that $\hat{x} = \hat{x}_1\| \ldots \|\hat{x}_\ell\|\hat{x}_*$. For this mapping we take $\ell = \lfloor |x|/n \rfloor$, $\hat{x}_i = x_{n(i-1)+1..ni}$ for $i \in [\ell]$, and $\hat{x}_* = x_{n\ell+1..|x|}$. For an incomplete block x, $\mathsf{pad}(x)$ denotes the complete block $x\|10^* = x\|1\|0^{n-|x|-1}$. For a complete block x and $k \in [n]$, $\mathsf{msb}_k(x)$ and $\mathsf{lsb}_k(x)$ denote the substring $x_{1..k}$ and $x_{n-k+1..n}$ respectively. For a binary string x, $\mathsf{ntz}(x)$ denotes the number of trailing zero bits in x.

RANDOM FUNCTIONS AND PERMUTATIONS — For a set $\mathcal{S}$, we write $x \leftarrow\!\$\, \mathcal{S}$ to denote that x is sampled from $\mathcal{S}$ uniformly. For a given domain $\mathcal{D}$ and a given co-domain $\mathcal{R}$, we denote by $\mathsf{Func}[\mathcal{D}, \mathcal{R}]$ the set of all functions from $\mathcal{D}$ into $\mathcal{R}$. We say Γ is an ideal random function from $\mathcal{D}$ to $\mathcal{R}$ to indicate that $\Gamma \leftarrow\!\$\, \mathsf{Func}[\mathcal{D}, \mathcal{R}]$. On a similar note, we denote by $\mathsf{Perm}[\mathcal{D}]$ the set of all permutations on $\mathcal{D}$. We say Π is an ideal random permutation on $\mathcal{D}$ to indicate that $\Pi \leftarrow\!\$\, \mathsf{Perm}[\mathcal{D}]$. For distinct inputs $x_1, \ldots, x_m \in \mathcal{D}$, $(\Gamma(x_1), \ldots, \Gamma(x_m)) \leftarrow\!\$\, \mathcal{R}^m$, and $(\Pi(x_1), \ldots, \Pi(x_m)) \leftarrow\!\$\, \mathcal{D}^{\underline{m}}$. In other words, an ideal random function Γ can be viewed as a *with-replacement* sampler from $\mathcal{R}$, whereas an ideal random permutation Π can be viewed as a *without-replacement* sampler from $\mathcal{D}$.

A. Distinguishers and Advantage

Given two oracles $\mathcal{O}_0$ and $\mathcal{O}_1$, an algorithm $\mathscr{A}$ that tries to distinguish between $\mathcal{O}_0$ and $\mathcal{O}_1$ is called a *distinguishing adversary* or a *distinguisher*. $\mathscr{A}$ plays an interactive game with $\mathcal{O}_b$ (for some bit b) in a black box fashion, and outputs a single bit at the end. For an oracle $\mathcal{O}$, $\mathscr{A}^{\mathcal{O}}$ denotes $\mathscr{A}$'s output after its interaction with $\mathcal{O}$, and we say that $\mathscr{A}$ wins the distinguishing game if $\mathscr{A}^{\mathcal{O}} = b$. The distinguishing advantage of $\mathscr{A}$ is defined as

$$\mathbf{Adv}_{\mathcal{O}_1;\mathcal{O}_0}(\mathscr{A}) := \left|\Pr\left[\mathscr{A}^{\mathcal{O}_1} = 1\right] - \Pr\left[\mathscr{A}^{\mathcal{O}_0} = 1\right]\right|.$$

Let $\mathbb{A}(q, \ell, \sigma, t)$ be the class of all distinguisher limited to q oracle queries, each of length at most ℓ blocks, total length $\sigma \le q\ell$, and t computations. We define

$$\mathbf{Adv}_{\mathcal{O}_1;\mathcal{O}_0}(q, \ell, \sigma, t) := \max_{\mathscr{A} \in \mathbb{A}(q,\ell,\sigma,t)} \mathbf{Adv}_{\mathcal{O}_1;\mathcal{O}_0}(\mathscr{A}).$$

NOTIONS OF VARIOUS ORACLES — $\mathcal{O}_0$ conventionally represents an ideal primitive, while $\mathcal{O}_1$ represents either an actual construction or a mode of operation built of some other ideal primitives. Typically the goal of the function represented by $\mathcal{O}_1$ is to emulate the ideal primitive represented by $\mathcal{O}_0$. We use the standard terms *ideal oracle* and *real oracle* for $\mathcal{O}_0$ and $\mathcal{O}_1$ respectively.

SECURITY GAME — In the context of this paper, a security game is a standard distinguishing game with an optional set of additional restrictions, chosen to reflect the desired security goal. For instance, some security games give bidirectional access to the underlying function of an oracle $\mathcal{O}$ and its inverse. This is denoted by $\mathcal{O}^{\pm}$. When we talk of distinguishing advantage with a specific security game G in mind, we include G in the superscript, e.g., $\mathbf{Adv}^{G}_{\mathcal{O}_1;\mathcal{O}_0}(\mathscr{A})$. Usually, $\mathcal{O}_0$ is completely identified by G, and hence dropped from the subscript.

B. PRF and PRP Security Games

Let $f : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ be a keyed function family from $\mathcal{D}$ to $\mathcal{R}$ indexed by the key space $\mathcal{K}$. In the pseudorandom function security game prf, the goal of any adversary $\mathscr{A}$ is to distinguish f from an ideal random function $\Gamma \leftarrow\!\$\, \mathsf{Func}[\mathcal{D}, \mathcal{R}]$. Hence the PRF-advantage of $\mathscr{A}$ against f is defined as

$$\mathbf{Adv}^{\mathsf{prf}}_{f}(\mathscr{A}) := \left|\Pr_{K \leftarrow\!\$\, \mathcal{K}}\left[\mathscr{A}^{f_K} = 1\right] - \Pr_{\Gamma}\left[\mathscr{A}^{\Gamma} = 1\right]\right|.$$

With q, ℓ, σ and t defined as above, we define the PRF-security of f against the class of adversaries $\mathbb{A}(q, \ell, \sigma, t)$ as $\mathbf{Adv}^{\mathsf{prf}}_{f}(q, \ell, \sigma, t)$. In the same way we can define PRP or pseudorandom permutation advantage. For a keyed permutation f over $\mathcal{R}$, we define

$$\mathbf{Adv}^{\mathsf{prp}}_{f}(\mathscr{A}) := \left|\Pr_{K \leftarrow\!\$\, \mathcal{K}}\left[\mathscr{A}^{f_K} = 1\right] - \Pr_{\Pi}\left[\mathscr{A}^{\Pi} = 1\right]\right|.$$

In a stronger variant of PRP game, called strong PRP (SPRP) security game, the adversary is given access to both the forward and inverse queries to the oracle. The SPRP advantage of $\mathscr{A}$ is defined analogously as $\mathbf{Adv}^{\mathsf{sprp}}_{f}(\mathscr{A})$.

C. Authenticated Encryption Security Game

A *nonce-based authenticated encryption scheme with associated data* (NAEAD) consists of a key space $\mathcal{K}$, a message space $\mathcal{M}$, an associated data space $\mathcal{A}$, a nonce space $\mathcal{N}$ and a tag space $\mathcal{T}$, along with two functions $\mathsf{Enc} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{M} \times \mathcal{T}$ and $\mathsf{Dec} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \times \mathcal{T} \to \mathcal{M} \cup \{\perp\}$, with the correctness condition that for any $K \in \mathcal{K}$, $N \in \mathcal{N}$, $A \in \mathcal{A}$, $M \in \mathcal{M}$, we have

$$\mathsf{Dec}(K, N, A, \mathsf{Enc}(K, N, A, M)) = M.$$

In addition, in most popular authenticated encryption schemes (including $\mathsf{OCB3}$), the map $\mathsf{proj}_{\mathcal{M}} \circ \mathsf{Enc}(K, N, A, \cdot)$ for fixed K, N, A is a length-preserving permutation, where $\mathsf{proj}_{\mathcal{M}} : \mathcal{M} \times \mathcal{T} \to \mathcal{M}$ is the projection on $\mathcal{M}$.

In the nonce-respecting authenticated encryption security game with associated data naead, $\mathcal{O}_1^+$ and $\mathcal{O}_1^-$ of the real

oracle are the encryption function $\mathsf{Enc}(K,\cdot,\cdot,\cdot)$ and decryption function $\mathsf{Dec}(K,\cdot,\cdot,\cdot,\cdot)$ respectively of the authenticated encryption scheme under consideration for a key K randomly chosen from $\mathcal{K}$; in the ideal oracle, $\mathcal{O}_0^+$ and $\mathcal{O}_0^-$ are $\Gamma \xleftarrow{\$} \mathsf{Func}[\mathcal{N}\times\mathcal{A}\times\mathcal{M},\mathcal{M}\times\mathcal{T}]$, and $\perp : \mathcal{N}\times\mathcal{A}\times\mathcal{M}\times\mathcal{T}\to\{\perp\}$ respectively. We henceforth refer to $(\Gamma,\perp)$ as the *ideal nonce-based authenticated encryption scheme*. The distinguishing adversary operates under the following restrictions — (a) no two encryption queries can have the same nonce, and (b) if an encryption query (N, A, M) yields (C, T), a decryption query (N, A, C, T) is not allowed.

SECURITY GOALS — Distinguishing advantage of any adversary $\mathscr{A}$ in naead with associated data will be denoted as $\mathbf{Adv}^{\mathsf{naead}}_{\mathcal{O}_1;\mathcal{O}_0}(\mathscr{A})$. Note that security under this formulation covers the two standard security goals of authenticated encryption:

- *Privacy* — Security against an adversary who tries to distinguish the AE or AEAD construction from an ideal random function Γ, and
- *Integrity* — Security against an adversary who tries to make a successful forging attempt on the scheme using decryption queries.

III. REVISITING SMALL-DOMAIN AXU HASH FUNCTIONS

An $(\mathcal{L},\mathcal{D},\mathcal{R})$-hash function $H : \mathcal{L}\times\mathcal{D}\to\mathcal{R}$ is a keyed function family from *domain or message space* $\mathcal{D}$ to *digest or hash space* $\mathcal{R}$ and indexed by the *key space* $\mathcal{L}$. In this work we fix $\mathcal{R}=\mathcal{B}$ and hence drop it from the notation.

Definition 1 (AXU hash function [38], [39])**.** An $(\mathcal{L},\mathcal{D})$-hash function H is called ϵ*-AXU* (almost XOR universal) hash function if $\forall\ X\neq Y\in\mathcal{D},\ \delta\in\mathcal{B}$,

$$\mathsf{DP}_H(X,Y:\delta) := \Pr_{L\xleftarrow{\$}\mathcal{L}}[H_L(X)\oplus H_L(Y)=\delta]\le\epsilon. \quad (1)$$

The term $\mathsf{DP}_H(X,Y:\delta)$ is also called the differential probability (or DP) of H on inputs X and Y and difference δ. The above definition says that the maximum differential probability (or MDP) of H is at most ϵ.

COMPOSITIONS — We consider two compositions of AXU hash functions:

- *Range Extension* — For ϵ_i-AXU hash functions $H_i:\mathcal{L}_i\times\mathcal{D}\to\mathcal{R}_i$, $i=1,2$,

$$H_1\|H_2 : (\mathcal{L}_1\times\mathcal{L}_2)\times\mathcal{D}\to\mathcal{R}_1\times\mathcal{R}_2$$
$$(L_1,L_2,X)\mapsto H_1(L_1,X)\|H_2(L_2,X)$$

- *Domain Extension* — For ϵ_i-AXU hash functions $H_i : \mathcal{L}_i\times\mathcal{D}_i\to\mathcal{R}$, $i=1,2$,

$$H_1\oplus H_2 : (\mathcal{L}_1\times\mathcal{L}_2)\times(\mathcal{D}_1\times\mathcal{D}_2)\to\mathcal{R}$$
$$((L_1,L_2),(X_1,X_2))\mapsto H_1(L_1,X_1)\oplus H_2(L_2,X_2)$$

It is easy to verify that in range extension, $H_1\|H_2$ is an $\epsilon_1\epsilon_2$-AXU hash function if L_1 and L_2 are independently sampled. It can also be verified that the domain extension hash $H_1\oplus H_2$ is a $\max\{\epsilon_1,\epsilon_2\}$-AXU hash function if L_1 is independent of L_2. We summarize the two observations in the following proposition.

TABLE I

FEATURE SUMMARY OF SOME SMALL-DOMAIN AXU HASH FUNCTION CANDIDATES, WHERE $\oplus,\odot,\ll,\|,\oplus_?$ DENOTE ADDITION, MULTIPLICATION, SHIFTS, CONCATENATION AND CONDITIONAL ADDITION OVER APPROPRIATE FIELDS; AESRD DENOTES AES ROUND FUNCTION, wt AND ntz DENOTES THE HAMMING WEIGHT AND TRAILING ZEROS FUNCTION

Hash Family	Sequential (operations)	Direct (operations)	Key Size	Domain Size in log base 2	MDP
xtimes [3]	$1\ll, 1\oplus_?$	$O(\lg i)\odot$	128	128	2^{-128}
gray [25], [9]	1 ntz(i), $1\oplus$	$1\oplus, 1\ll, 1\odot$	128	128	2^{-128}
mtrx [26], [27]	$2\ll, 1\oplus$ [#]	$O(\lg i)\odot$	128	128	2^{-128}
mlin	wt$(i)\oplus$	wt$(i)\oplus$	$128r$	r	2^{-128}
clh [29]	wt$(i)\ll$, wt$(i)\oplus$	wt$(i)\ll$, wt$(i)\oplus$	128	127	2^{-128}
sts [9]	$1\|, 1\oplus, 3\ll$	$1\|, 1\oplus, 3\ll$	128	6	2^{-128}
lcube	$1\oplus, 2\odot$	$1\oplus, 2\odot$	128	128	2^{-128}
linv [3]	$1\oplus, 1\odot$	$1\oplus, 1\odot$	128	128	2^{-128}
aes4 [35]	4 AESRD	4 AESRD	512	128	2^{-113}
aes2 (Section IV)	2 AESRD	2 AESRD	256	32	2^{-113}
aes1 (Section IV)	1 AESRD	1 AESRD	128	8	2^{-96}

[#] Note that [26] defines many three-operation maskings for 128 bit binary field.

Proposition 1. *Let H_i be ϵ_i-AXU hash functions for $i\in\{1,2\}$. Then,*

- *$H_1\|H_2$ hash function is an $\epsilon_1\epsilon_2$-AXU hash function; and*
- *$H_1\oplus H_2$ hash function is a $\max\{\epsilon_1,\epsilon_2\}$-AXU hash function.*

A. Examples of Small-Domain AXU Hash Function

Symmetric-key cryptographic literature is rich in AXU hash function candidates. In this work we concentrate only on *small-domain* hash functions that can be used to generate masks in OCB. In what follows, we discuss some examples of AXU hash functions, with a summary as in Table I.

Example 1 (α-multiplication based hash [3])**.** For some reasonably large $\ell\in[2^n-1]$, let $\mathcal{D}=[\ell]$ and $\mathcal{L}=\mathcal{B}$. The hash function xtimes, defined as $\forall L\in\mathcal{B}, i\in[\ell]$,

$$\mathsf{xtimes}_L(i)=\mathsf{xtimes}(L,i):=\alpha^i\odot L, \quad (2)$$

is a 2^{-n}-AXU hash. xtimes has been used to generate position dependent input masks in many symmetric-key designs including OCB designs [3], [8], [9], OTR [40], PMAC [25], PMAC_Plus [41], EME [42], COPA [11] and ELmD [12]. The popularity of xtimes is mainly due to its efficiency in sequential computation, as we have

$$\forall i\ge 1,\ L\in\mathcal{B},\ \mathsf{xtimes}_L(i+1)=\alpha\odot\mathsf{xtimes}_L(i),$$

and α-multiplication is efficient (one shift and one conditional bit-wise XOR operation). While the sequential computation of $\mathsf{xtimes}_L(i)$ is very efficient, the same is not reflected in direct computation of $\mathsf{xtimes}_L(i)$ when i is a large integer. In general it may require $O(\lg i)$ many field multiplications. Even when the α^i values are precomputed, we need one field multiplication which is not as efficient as one α-multiplication. Indeed we can even avoid the precomputation and simply use $i\odot L$ as the hash definition.

Example 2 (Gray-code based hash [9], [25])**.** For reasonably large $\ell\in[2^n-1]$, let $\mathcal{D}=[\ell]$ and $\mathcal{L}=\mathcal{B}$. The hash function

gray is defined as,

$$\forall L \in \mathcal{B},\ i \in [\ell], \quad \mathsf{gray}_L(i) = \mathsf{gray}(L, i) := \gamma(i) \odot L \quad (3)$$

where $\gamma(i) := i \oplus (i \gg 1)$ is the gray code value for i. The Gray code sequence has a very nice feature that for $i \geq 1$, the i-th gray code can be written in terms of the $(i-1)$-th gray code, i.e

$$\gamma(i) = \gamma(i-1) \oplus 2^{\mathsf{ntz}(i)},$$

When we substitute this in (3) we get an alternate definition $\forall L \in \mathcal{B},\ i \in [\ell]$,

$$\mathsf{gray}_L(i) = \mathsf{gray}(L, i) := \oplus_{j=1}^{i} L_{\mathsf{ntz}(j)}. \quad (4)$$

where $L_j = \alpha^j \odot L$ for all $j < n$. In [9] and RFC7253 [43], gray has been shown to be a 2^{-n} AXU hash function. Note that we have

$$\forall i \geq 1,\ L \in \mathcal{B},\ \mathsf{gray}_L(i+1) = \mathsf{gray}_L(i) \oplus L_{\mathsf{ntz}(i+1)}.$$

So if the L_j values are precomputed, then in sequential computation gray hash only requires, ntz computation along with a single n-bit XOR operation. Hence it is very efficient in sequential computations. This hash function is applied in OCB3 [9] (RFC7253 [43]) and PMAC [25].

Example 3 (Matrix-powered hash [26], [27])**.** Let $n = wn'$ for some positive integer w, called word size, whose values are generally software friendly such as 8, 16, 32, 64 etc. Let $\mathcal{W} = \{0,1\}^w$ and $\mathcal{L} = \mathcal{W}^{n'}$ and $\mathcal{D} = [\ell]$ for some reasonably large $\ell \in [2^n - 1]$. Let M be an invertible $n' \times n'$ matrix whose entries are from the field[2] $\mathcal{W}$ such that for all $i \in [2^n - 1]$, $I_{n'} + M^i$ is invertible. Then the hash function mtrx (called *matrix-powered hash*), defined as

$$\forall L \in \mathcal{W}^{n'}, i \in [\ell], \quad \mathsf{mtrx}_L(i) = \mathsf{mtrx}(L, i) := M^i \cdot L, \quad (5)$$

is a 2^{-n}-AXU. Note here that L is viewed as a vector over $\mathcal{W}$. Various matrix-powered hash candidates are given in [26] and [27]. But, similar to xtimes hash, all those hash functions are efficient in sequential computation and inefficient in direct computation.

The matrix-powered hash function is a very general candidate. For example, xtimes and gray hash functions can be viewed as instances of matrix-powered hash for appropriate choices of the matrix M. The readers are directed to [26] and [27] for a detailed exposition on matrix-powered hash functions. A table of various word-oriented matrices has been listed in [26].

Example 4. Some other finite field based constructions, with $\mathcal{L} = \mathcal{D} = \mathcal{B}$, are the following: $\forall L \in \mathcal{B},\ x \in \mathcal{B}$

- $\mathsf{lcube}_L(x) = (L \oplus x)^3$;
- $\mathsf{linv}_L(x) = (L \oplus x)^{-1}$ (if $x \neq L$), 0 otherwise.

It is easy to verify that lcube is 2^{1-n}-AXU and linv is 2^{2-n}-AXU hash function.

[2] $\{0,1\}^m$ can be viewed as the binary field by fixing a deg m primitive polynomial.

Example 5 (Multi-linear hash)**.** For $\ell \in [2^n - 1]$, let $\mathcal{D} = [0..\ell]$ and $\mathcal{L} = \mathcal{B}^{\lg \ell}$. The hash function mlin (called multi-linear hash), is defined as

$$\forall L \in \mathcal{L},\ b \in \mathcal{D}, \quad \mathsf{mlin}_L(b) = \mathsf{mlin}(L, b) := \oplus_{i=1}^{r} b_i L_i, \quad (6)$$

is a 2^{-n}-AXU hash function, where $L_1, \ldots, L_r \in \mathcal{B}$ and $b_1, \ldots, b_r \in \{0,1\}$. In Grain-128a [44], the tag is computed using mlin, where the keys are derived using Toeplitz matrix.

Example 6 (Circulant hash [29])**.** In [29] Minematsu presented a small-domain hash function based on data-dependent rotations. For $\ell \in [2^{n-1} - 1]$, let $\mathcal{D} = [0..\ell]$ and $\mathcal{L} = \mathcal{B}$. The hash function clh (called circulant hash) is defined as $\forall K \in \mathcal{L},\ x \in \mathcal{D}$,

$$\mathsf{clh}_L(x) = \mathsf{clh}(L, x) := \bigoplus_{i \in [\lg \ell]:\ x_i = 1} (L \ll (i-1)). \quad (7)$$

For direct computation, the hash function might require $n-1$ rotations and $n-2$ XORs in the worst case; for sequential computation some optimization is possible for the i-th input given the $(i-1)$-th output. In [29], clh is shown to be both 2^{-128}-AXU and 2^{-128}-uniform hash function.

Example 7 (Stretch-then-shift hash [9])**.** In [9] Krovetz and Rogaway presented a highly efficient AXU hash function over a small domain $\mathcal{D} = [64]$. It follows the general data-dependent rotation technique by Minematsu [29]. Fix $c \in \mathbb{N}^+$. For $\mathcal{D} = [0..\psi(c) - 1]$ and $\mathcal{L} = \mathcal{B}$, where $\psi(c)$ is a positive integer denoting the maximum domain size for a fixed c. The hash function sts (called stretch-then-shift hash) is defined as $\forall L \in \mathcal{B},\ i \in [0..\psi(c) - 1]$,

$$\mathsf{sts}_L(i) = \mathsf{sts}(L, i) := \mathsf{msb}_n(\mathsf{stretch}_c(L) \ll i), \quad (8)$$

where $\forall L \in \mathcal{B}, \quad \mathsf{stretch}_c(L) := L \| (L \oplus (L \ll c))$. The hash function first stretches the key based on a parameter c, and then makes data-dependent shift before outputting the most significant n bits; hence the name. For $n = 128$, $c = 8$ and $\psi(c) = 64$, Krovetz and Rogaway showed that sts achieves 2^{-128}-AXU and 2^{-128}-uniform bound [9].

IV. AES-based Small-Domain AXU Hash Functions

In the previous section we listed some hash functions based on finite field operations and data dependent rotations. Now we will propose some hash functions based on the differential properties of the AES rounds. Before moving forward, we give the standard and well-known [30], [35] definition for differential probability of a (keyed) permutation.

Definition 2. Let π be a permutation over $\mathbb{F}_{2^n}$ and Π_K be a keyed permutation over $\mathbb{F}_{2^n}$ with key $K \leftarrow\!\$\, \mathcal{K}$. For any given non-zero $a, b \in \mathbb{F}_{2^n}$ the differential probability of π, is defined as

$$\mathsf{dp}_\pi(a, b) := \Pr_X[\pi(X) \oplus \pi(X \oplus a) = b].$$

The maximum differential probability (MDP) of π is defined as

$$\mathsf{mdp}(\pi) := \max_{a \neq 0, b} \mathsf{dp}_\pi(a, b).$$

Similarly, the expected differential probability (EDP) of Π_K, is defined as

$$\begin{aligned}&\mathsf{edp}_{\Pi_K}(a,b)\\&:= \Pr_{K,X}[\Pi_K(X)\oplus\Pi_K(X\oplus a)=b]\\&=\sum_{K\in\mathcal{K}}\Pr_X[\Pi_k(X)\oplus\Pi_k(X\oplus a)=b\mid k]\cdot\Pr[K=k]\\&=\sum_{K\in\mathcal{K}}\mathsf{dp}_{\Pi_k}(a,b)\cdot\Pr[K=k]=\mathbb{E}_K[\mathsf{dp}_{\Pi_K}(a,b)],\end{aligned}$$

where $\mathbb{E}_K$ denotes the expectation over K. The maximum expected differential probability (MEDP) of Π_K is defined as

$$\mathsf{medp}(\Pi_K) := \max_{a\neq 0,b}\mathsf{edp}_{\Pi_K}(a,b).$$

Note that we have also used the term differential probability in the context of AXU hash functions in Definition 1. Now that we have formally defined the differential probability of a (keyed) permutation, it is imperative that we discuss the similarity and differences between the two probabilities. The major difference is the source of randomness. In the AXU view of differential probability, the randomness is due to the hash key, whereas here one of the input is chosen randomly. Even though this seems to be a major difference, but for certain class of functions the two views have a very nice relationship. Suppose π is a (possibly keyed) permutation over $\mathbb{F}_{2^n}$ with keyspace $\mathcal{K}$ (empty set when π is keyless). We define a function $\pi' : \mathbb{F}_{2^n}\times\mathcal{K}\times\mathbb{F}_{2^n}\to\mathbb{F}_{2^n}$ as a keyed family of function indexed by $K, K'\in\mathbb{F}_{2^n}\times\mathcal{K}$, such that

$$\forall K, K', x\in\mathbb{F}_{2^n}\times\mathcal{K}\times\mathbb{F}_{2^n},\quad \pi'_{K,K'}(x) := \pi_{K'}(K\oplus x).$$

Lemma 1 is a well-known result that we prove here just for the sake of completeness.

Lemma 1. *For some $\epsilon\geq 0$,*

1) *If π is keyless and $\mathsf{mdp}(\pi)=\epsilon$ then π' is an ϵ-AXU hash for $K\leftarrow_{\$}\mathbb{F}_{2^n}$.*
2) *If π is keyed and $\mathsf{medp}(\pi)=\epsilon$ then π' is an ϵ-AXU hash for $K\leftarrow_{\$}\mathbb{F}_{2^n}$ and $K'\leftarrow_{\$}\mathcal{K}$.*

Proof: We prove the unkeyed version of the lemma. The keyed version can be proved in similar fashion. Let $x, x'\in\mathbb{F}_{2^n}$ such that $x\neq x'$ and $\delta\in\mathbb{F}_{2^n}$. If $\delta=0$ then the result is vacuously true as π' is a permutation. Suppose $\delta\neq 0$. Then, we have

$$\begin{aligned}\mathsf{DP}_{\pi'}(x,x'\ :\ \delta)&=\Pr_K[\pi'(K\oplus x)\oplus\pi'(K\oplus x')=\delta]\\&=\Pr_X[\pi'(X)\oplus\pi'(X\oplus x'\oplus x)=\delta]\\&\leq\mathsf{dp}(\pi)=\epsilon.\end{aligned}$$

□

A. Revisiting MEDP Bounds for the AES

The AES block cipher is a Substitution Permutation Network (SPN) with block size n = 128, internal s-box input size b = 8, where all s-boxes are identical, denoted by S. The permutation layer, denoted by ϕ, consists of a bytewise permutation followed by four identical 32-bit linear transformations applied in parallel. One round of AES, denoted by 1-AESRD (without the subkey mixing) is nothing but $\phi\circ S$. Using the 1-AESRD round function, we can define the keyed function i-AESRD for all integers $i>1$ as follows: $\forall x\in\mathbb{F}_{2^{128}}, K\in\mathbb{F}_{2^{128}}^{i-1}$,
For $i=2$:

$$2\text{-AESRD}_K(x) = 1\text{-AESRD}(K_1\oplus 1\text{-AESRD}(x))$$

For $i>2$:

$$i\text{-AESRD}_K(x) = 1\text{-AESRD}(K_{i-1}\oplus(i-1)\text{-AESRD}_{K_{[i-2]}}(x))$$

The differential properties of AES has been a well-studied [30]–[32], [45], [46] topic in symmetric-key cryptography. In fact, one of the design criteria for AES [45], [47] was protection against linear [48] and differential cryptanalysis [49].

In [46], Daemen and Rijmen first observed that due to the specific nature of the AES permutation layer ϕ, the differential probabilities over 2 AES rounds are equivalent to those over a reduced SPN structure, which they called a *super box*. Basically the two rounds of AES over 128-bit input can be viewed as four parallel invocations of the super box with independent keys (disjoint substrings of a uniform string are independent) each working with distinct 32 bits of the input. We denote an AES super box instantiated with a subkey K by $\texttt{aesSB}_K$.

Keliher and Sui [31], [32] later used this idea to derive a tight MEDP bound for 2-AESRD. Specifically, in [31, Theorem 1, Theorem 2] and [32, section 4.1] they proved that the MEDP for AES super box is 1.656×2^{-29}, which is equivalent to the MEDP for 2-AESRD. We refer the readers to [32], [46] for further exposition on AES super box and its relation with 2-AESRD. Using the fact that the upper bound on the MEDP for 4 or more rounds of AES is equal to the 4^{th} power of the upper bound on the MEDP for 2-AESRD, Keliher and Sui [32] further showed that MEDP for t-AESRD for $t\geq 4$ is upper bounded by 1.881×2^{-114}. We summarize these results in Proposition 2.

Proposition 2. *[32, section 4.1] Let X_1, X_2, X_3 be uniformly and independently sampled from $\mathbb{F}_{2^{128}}$ and $Y\leftarrow_{\$}\mathbb{F}_{2^{32}}$. Then,*

1) $\mathsf{medp}(\texttt{aesSB}_Y)\approx 1.656\times 2^{-29}$.
2) $\mathsf{medp}(2\text{-AESRD}_{X_1})\approx 1.656\times 2^{-29}$.
3) $\mathsf{medp}(4\text{-AESRD}_{X_1,X_2,X_3})\leq 1.881\times 2^{-114}$.

B. Our AES-Based Proposals

In the following discussion we fix $n=128$, i.e. $\mathcal{B}=\{0,1\}^{128}$. Before presenting our proposals we start with a simple 4-AESRD based hash function in the following example.

Example 8. Let $\mathcal{D}=\mathcal{B}$ and $\mathcal{L}=\mathcal{B}^4$. For $L=(L_1,L_2,L_3,L_4)\in\mathcal{B}^4$ and $x\in\mathcal{B}$, the 4-AESRD based hash function aes4 is defined as

$$\mathsf{aes4}_L(x)=\mathsf{aes4}(L,x) := 4\text{-AESRD}_{L_2,L_3,L_4}(L_1\oplus x).$$

From Lemma 1 and Proposition 2, it is straightforward to see that aes4 is a 1.88×2^{-114}-AXU hash function. In [35] Minematsu and Tsunoo used a variant of aes4 to construct

a universal hash function and subsequently extended it to construct a MAC. Jakimoski and Subbalakshmi [50] further built upon their work to get more efficient hash functions and MAC.

1) 1-Round AES Based Hash Function: The MDP for AES s-box S is known to be 2^{-6} [30]. We define a keyed function $S' : \mathbb{F}_{2^8} \times \mathbb{F}_{2^8} \to \mathbb{F}_{2^8}$ as

$$\forall K, x \in \mathbb{F}_{2^8}^2, \quad S'_K(x) := S(K \oplus x).$$

Clearly S and S' satisfy the conditions in Lemma 1, hence S' is 2^{-6}-AXU hash function. Now, let $\mathcal{D} = \{0,1\}^8$ (viewed as [256]) and $\mathcal{L} = \mathcal{B}$ (viewed as $\mathcal{D}^{16}$). Given a key L, and an input $i \in \mathcal{D}$, we define the one round AES hash, aes1 as

$$\begin{aligned}\mathsf{aes1}_L(i) = \mathsf{aes1}(L, i) &:= \mathsf{1\text{-}AESRD}(L \oplus i^{16}) \\ &= \phi(S'_{L_1}(i) \| \cdots \| S'_{L_{16}}(i)). \end{aligned} \tag{9}$$

Recall that ϕ is the linear transformation or permutation layer used in AES. It is well-known that the AXU property is preserved under linear composition over an AXU has function. Thus ϕ being a linear transformation does not affect the AXU bound for aes1 and can be ignored. When L is uniformly drawn from $\mathcal{L}$ and viewed as an element of $\mathcal{D}^{16}$, then L_i's are mutually independent. So by repeatedly using Proposition 1 and Lemma 1, we get the AXU bound for aes1 in Lemma 2.

Lemma 2. *aes1 as defined above is 2^{-96}-AXU hash function.*

2) 2-Round AES Based Hash Function: From Proposition 2 we already now that the MDP for aesSB is upper bounded by 1.656×2^{-29}. Again we define a keyed function $\mathtt{aesSB}' : \mathbb{F}_{2^{32}} \times \mathbb{F}_{2^{32}} \to \mathbb{F}_{2^{32}}$ as

$$\forall K, x \in \mathbb{F}_{2^{32}}^2, \quad \mathtt{aesSB}'_K(x) := \mathtt{aesSB}_{K_2}(K_1 \oplus x).$$

Clearly aesSB′ is 1.656×2^{-29}-AXU hash function (using Lemma 1). Now let $\mathcal{D} = \{0,1\}^{32}$ (viewed as $[2^{32}]$) and $\mathcal{L} = \mathcal{B} \times \mathcal{B}$ (viewed as $\mathcal{D}^4 \times \mathcal{D}^4$). Given a key $L = (L, L') \in \mathcal{D}^4 \times \mathcal{D}^4$, and an input $i \in \mathcal{D}$, we define the two round AES hash, aes2 as

$$\begin{aligned}\mathsf{aes2}_{L,L'}(i) = \mathsf{aes2}(L, L', i) &:= \mathsf{2\text{-}AESRD}_{L'}(L \oplus i^4)) \\ &= \phi(\mathtt{aesSB}'_{L'_1}(L_1 \oplus i) \| \cdots \| \mathtt{aesSB}'_{L'_4}(L_4 \oplus i)),\end{aligned}$$

where each L_i / L'_i is a 32-bit strings. Using a similar line of argument as used in case of aes1 we get the AXU bound for aes2 in Lemma 3.

Lemma 3. *aes2 as defined above is 1.881×2^{-114}-AXU hash function.*

So we see that aes1 and aes2 are sub-optimum, but decent AXU hashes given the domains they are applied upon. In fact it is quite surprising that this fact was not discovered yet. Now based on the existing proof techniques, the straightforward application of these hash functions in OCB is not advisable as they give sub-optimum security bounds.

V. Generic View of OCB

In previous two sections we have seen various examples of AXU hashes over small domains $[\ell]$, where ℓ is typically a large positive integer in $[2^n]$. These hash functions are useful when the goal is to embed position-based dependency (called masks) to the input blocks of the underlying primitive, typically a block cipher. This type of embedding is used in many encryption, MAC and AE algorithms. Some prominent schemes among these belong to the OCB like design paradigms. We call the embedding function — *mask-generating function* (MGF).

Mask Generating Function — Typically, an MGF, $\lambda : \mathcal{L} \times \mathcal{N} \times \mathbb{N}^+ \to \mathcal{B}$ is a $(\mathcal{L}, \mathcal{N} \times \mathbb{N}^+)$-hash function family indexed by the key space $\mathcal{L}$. Here $\mathcal{N}$ is typically called the nonce space. For notational simplicity we will sometime write $\lambda(i) = \lambda_L(N, i)$, i.e $N \in \mathcal{N}$ and $L \in \mathcal{L}$ will be clear from the context. Next we define generic abstraction for OCB, called GOCB, based on $\Pi \leftarrow\!\!\$\ \mathsf{Perm}[\mathcal{B}]$ and λ. Figure 1 illustrates the schematic view of the encryption-decryption algorithms for GOCB and the complete algorithmic descriptions is given in Algorithm 1. For $i \in \mathbb{N}^+$, we will refer M_i, A_i and C_i as message block, associated data block and ciphertext block. For $i \in \mathbb{N}^+$, we will refer U_i and X_i as input blocks; V_i and Y_i as output blocks; and S^i as final block. In GOCB $X_\oplus$ and $Y_\oplus$ will be referred as the tag input and output block respectively. We will use the term GPHash[3] for the associated data processing phase of GOCB.

Note that the constructions employ different MGFs for special blocks (last block, checksum block etc.) processing. Usually for concrete constructions these MGFs are simple variants of the usual MGF λ. Similarly the key for the MGF is usually derived from the underlying random permutation. For the sake of simplicity we will assume that the MGF key is drawn independently, and the MGFs for special blocks are also defined independently.

In this paper we only focus on the mask generation component and its properties. This is mainly because GOCB greatly resemble simple variant of ECB mode when the masking is ignored. Hence both efficiency and security mostly rely on the mask-generation phase, albeit the two issues are mutually orthogonal. We discuss the two issues separately starting with efficiency.

Mask-generation and Efficiency. As noted before, if we ignore the input masking, GOCB is simply a variant of ECB, which is arguably the most efficient cryptographic mode of operation, both in hardware and software. So the MGF should preferably have the following properties:

- *Sequential Computation* — MGF should be fast enough to compute the mask outputs for consecutive inputs. More formally, for some r and for any i, $\lambda(i), \lambda(i+1), \ldots, \lambda(i+r-1)$ should be efficiently computable. Here r can be viewed as a parameter. One may observe the performance for different choices of r and then choose the best possible one.
- *Direct Computation* — MGF should be easily computable on any arbitrary input. In other words, for all i, $\lambda(i)$

[3] PHash is a commonly used terminology for the hash component in OCB.

Algorithm 1 GOCB Authenticated Encryption, Where A_*, M_*, C_* Are the Incomplete (Possibly Empty) Blocks at the End of A, M, C, Respectively; λ_1 Is a Keyed (i.e. the Key Is Implicit) Mask-Generating Function, and λ_i for $1 < i \leq 6$ Are Minor Variants of λ_1, Deployed in Various Boundary Conditions, Like Incomplete Block Processing, Tag Generation etc.

function GOCB$[\Pi, \lambda]$.Enc(N, A, M)
$S \leftarrow 0^n$
$M_\oplus \leftarrow 0^n$
$(A_1, \ldots, A_m, A_*) \xleftarrow{n} A$
$(M_1, \ldots, M_\ell, M_*) \xleftarrow{n} M$
for $i \leftarrow 1$ **to** m **do**
  $U_i \leftarrow A_i \oplus \lambda_5(N, i)$
  $V_i \leftarrow \Pi(U_i)$
  $S \leftarrow S \oplus V_i$
if $|A_*| > 0$ **then**
  $U_* \leftarrow \mathsf{pad}(A_*) \oplus \lambda_6(N, m)$
  $V_* \leftarrow \Pi(U_*)$
  $S \leftarrow S \oplus V_*$
for $i \leftarrow 1$ **to** ℓ **do**
  $M_\oplus \leftarrow M_\oplus \oplus M_i$
  $X_i \leftarrow M_i \oplus \lambda_1(N, \ell)$
  $Y_i \leftarrow \Pi(X_i)$
  $C_i \leftarrow Y_i \oplus \lambda_1(N, i)$
if $|M_*| > 0$ **then**
  $M_\oplus \leftarrow M_\oplus \oplus \mathsf{pad}(M_*)$
  $X_* \leftarrow \lambda_2(N, \ell)$
  $Y_* \leftarrow \Pi(X_*)$
  $\overline{C_*} \leftarrow Y_* \oplus \mathsf{pad}(M_*)$
  $C_* \leftarrow \mathsf{msb}_{|M_*|}(\overline{C_*})$
  $C \leftarrow (C_1, \ldots, C_\ell, C_*)$
  $X_\oplus \leftarrow M_\oplus \oplus \lambda_4(N, \ell)$
  $Y_\oplus \leftarrow \Pi(X_\oplus)$
  $T \leftarrow Y_\oplus \oplus \lambda_4(N, \ell) \oplus S$
else
  $C \leftarrow (C_1, \ldots, C_\ell)$
  $X_\oplus \leftarrow M_\oplus \oplus \lambda_3(N, \ell)$
  $Y_\oplus \leftarrow \Pi(X_\oplus)$
  $T \leftarrow Y_\oplus \oplus \lambda_3(N, \ell) \oplus S$
return (C, T)

function GOCB$[\Pi, \lambda]$.Dec(N, A, C, T)
$S \leftarrow 0^n$
$M_\oplus \leftarrow 0^n$
$(A_1, \ldots, A_m, A_*) \xleftarrow{n} A$
$(C_1, \ldots, C_\ell, C_*) \xleftarrow{n} C$
for $i \leftarrow 1$ **to** m **do**
  $U_i \leftarrow A_i \oplus \lambda_5(N, i)$
  $V_i \leftarrow \Pi(U_i)$
  $S \leftarrow S \oplus V_i$
if $|A_*| > 0$ **then**
  $U_* \leftarrow \mathsf{pad}(A_*) \oplus \lambda_6(N, m)$
  $V_* \leftarrow \Pi(U_*)$
  $S \leftarrow S \oplus V_*$
for $i \leftarrow 1$ **to** ℓ **do**
  $Y_i \leftarrow C_i \oplus \lambda_1(N, \ell)$
  $X_i \leftarrow \Pi^{-1}(Y_i)$
  $M_i \leftarrow X_i \oplus \lambda_1(N, i)$
  $M_\oplus \leftarrow M_\oplus \oplus M_i$
if $|C_*| > 0$ **then**
  $X_* \leftarrow \lambda_2(N, \ell)$
  $Y_* \leftarrow \Pi(X_*)$
  $M_* \leftarrow \mathsf{msb}_{|C_*|}(Y_*) \oplus C_*$
  $M \leftarrow (M_1, \ldots, M_\ell, M_*)$
  $M_\oplus \leftarrow M_\oplus \oplus \mathsf{pad}(M_*)$
  $X_\oplus \leftarrow M_\oplus \oplus \lambda_4(N, \ell)$
  $Y_\oplus \leftarrow \Pi(X_\oplus)$
  $T' \leftarrow Y_\oplus \oplus S$
else
  $M \leftarrow (M_1, \ldots, M_\ell)$
  $X_\oplus \leftarrow M_\oplus \oplus \lambda_3(N, \ell)$
  $Y_\oplus \leftarrow \Pi(X_\oplus)$
  $T' \leftarrow Y_\oplus \oplus S$
if $T' = T$ **then**
  return M
else
  return $\perp$

should be efficiently computable without the knowledge of any other mask outputs. This is the usual constraint for hash functions. Direct computation of masks is an important feature as it leads to features like random read access and massive parallelism. Given large number of parallel processing units (multiple CPU cores or GPUs) GOCB is completely parallel when the underlying MGF is efficient in direct computation. This holds even when the message length is not known beforehand, a scenario that is common in data streams.

- *Constant-time Computation* — A secondary requirement for any MGF is constant computation time for all inputs. While this is a secondary requirement as far as efficiency is concerned, it helps in mitigating a form of side channel attacks called timing attack [33], [51].

MASK-GENERATION AND SECURITY — From the security point of view, earlier works [3], [9], [37] required strong (almost 2^{-n}) AXU and 1-universal (regular, i.e $\Pr[H(x) = y] = O(2^{-n})$) property (some time implicitly) from the underlying MGFs. Since allmost all of the hash functions discussed in this work are regular, we will implicitly assume that MGF has this property. We will focus on the more dominant, AXU property. In this setting the advantage is generally bounded in terms of ϵ, the AXU bound of MGF. For instance the bound for OCB is roughly $O(\sigma^2\epsilon)$ respectively; hence the need for $O(2^{-n})$-AXU bound. Naturally, both aes1 and aes2 would suffer significant security loss in the existing setting. This is probably one of the reasons, these hash functions have not seen much attention. Yet another reason is the fact that aes1 and aes2 are defined over restricted domains $\{0,1\}^8$ and $\{0,1\}^{32}$ respectively.

A. Locally-Imperfect XOR Universal Hash Functions

The immediate question in light of the above discussion is as follows:

> *Can we relax the AXU condition on MGFs to use efficient directly computable functions like* aes1 *and* aes2 *, maintaining comparable security?*

We answer this in affirmative by proposing a slightly different universal notion — *Locally-Imperfect XOR Universal (or LIXU) Hash Functions*. The idea is to model a hash function H whose differential probability depends on the grouping of the given inputs. More precisely, we partition the inputs in several disjoint subsets with bounded cardinality, say at most r.[4]

[4]In fact we try to minimize the value of r.

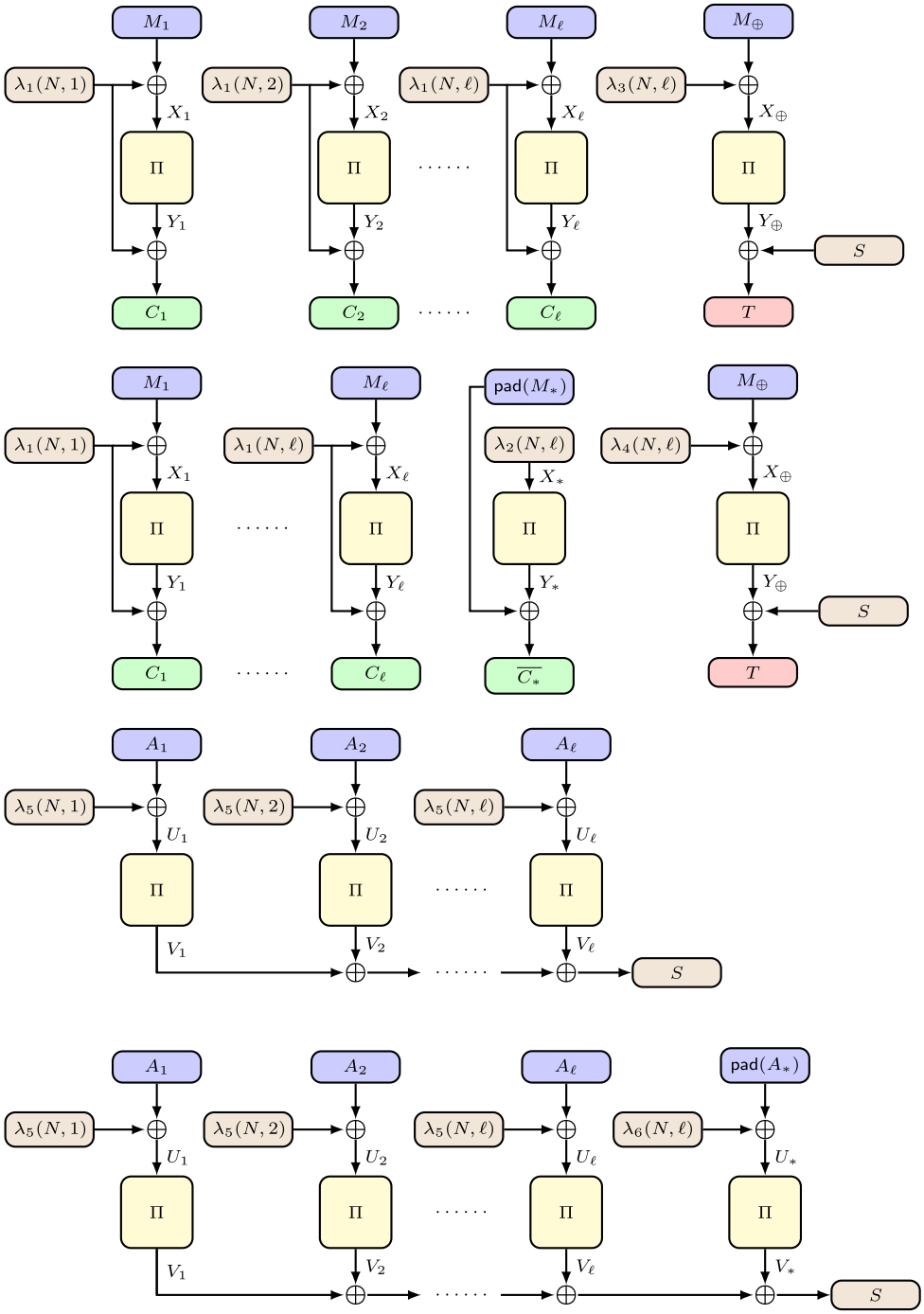

Fig. 1. Schematic diagram of GOCB— **Top to Bottom:** encrypting M when $|M_*| = 0$; encrypting M when $|M_*| > 0$; hashing A when $|A_*| = 0$; hashing A when $|A_*| > 0$.

Now the additional property introduced by LIXU is as follows: for two inputs say x and y from distinct partitions the differential probability is at most 2^{-n}, and the probability degrades to $\epsilon \geq 2^{-n}$, when x and y belong to the same partition. Formally we define the notion of Locally-Imperfect XOR Universal in Definition 3.

Definition 3 (Locally-Imperfect XOR Universal Hash Function)**.** Let $H : \mathcal{L} \times \mathcal{D} \to \mathcal{B}$ be a $(\mathcal{L}, \mathcal{D})$-hash function family indexed by the keyspace $\mathcal{L}$. For a fixed $r \in \mathbb{N}^+$ and $\epsilon \geq 2^{-n}$, we say that H is an (ϵ, r)-locally-imperfect XOR universal, or LIXU, hash function if and only if there exists a partitioning $\mathcal{D} = \mathcal{P}_1 \sqcup \mathcal{P}_2 \sqcup \cdots \sqcup \mathcal{P}_k$ with $k \geq 2$, such that $|\mathcal{P}_i| \leq r$ for $i \in [k]$, and for distinct $x, y \in \mathcal{D}$, and $\delta \in \mathcal{B}$,

$$\Pr_{L \xleftarrow{\$} \mathcal{L}}[H_L(x) + H_L(y) = \delta] \leq \begin{cases} \epsilon & \text{if } x, y \in \mathcal{P}_i, \\ \frac{1}{2^n} & \text{if } x \in \mathcal{P}_i,\ y \in \mathcal{P}_j, i \neq j. \end{cases}$$

Here r is called the *width* of H. We say that $x, y \in \mathcal{D}$ are *local pairs* (to each other and as a pair as well) if and only if

they belong to the same partition. So a LIXU hash function behaves as a perfect XOR universal hash function when the inputs are not local pairs, and behaves as an imperfect (or almost) XOR universal hash function when the inputs are local pairs. Note that an (ϵ, r)-LIXU hash implies an ϵ-AXU hash, but the vice-versa may not be true. Further it is rather trivial to establish that any 2^{-n}-AXU hash function over the domain $\mathcal{D}$ is also a $(2^{-n}, r)$-LIXU hash function for all $r \leq |\mathcal{D}| - 1$. We note this simple fact in Proposition 3.

Proposition 3. *A 2^{-n}-AXU hash H is a $(2^{-n}, r)$-LIXU hash for all $r \leq |\mathcal{D}| - 1$, where $\mathcal{D}$ is the domain of H. Notably, H is a $(2^{-n}, 1)$-LIXU hash function.*

In this paper, we use a specific type of LIXU hash functions defined over the domain $\mathcal{N} \times \mathbb{N}^+$, where the domain is partitioned as follows: $(N, i) \neq (N', i') \in \mathcal{N} \times \mathbb{N}^+$ are said to be local pairs if and only if $N = N'$ and $\lceil i/r \rceil = \lceil i'/r \rceil$. Here, $\mathcal{N}$ denotes the nonce space. We sometime use chunk and partition interchangeably, and for any $i \in \mathbb{N}^+$, we call $\lceil i/r \rceil$ the chunk number of i.

B. LIXU Hash as MGF in GOCB

Previous works on OCB have assumed that the underlying MGF is an AXU hash. We depart from this setting and employ LIXU hash as MGF. Since LIXU and AXU are not exactly compatible, we will require a slightly different analysis. Particularly the TBC based abstraction [8] for OCB is not useful here. We now present the main security result of the paper, along with an overview of our proof approach. The formal proof is postponed to Section VI.

Theorem 1 (GOCB NAEAD Bound)**.** *For a fixed $r \leq \ell \in \mathbb{N}^+$ and $\epsilon \geq 0$, let λ be an (ϵ, r)-LIXU hash function over $\mathcal{N} \times \mathbb{N}^+$. Let $\mathbb{A}(q, \tilde{q}, \ell, \sigma, \tilde{\sigma})$ be the class of all adversaries that make q encryption queries consisting of μ message blocks in all with at most ℓ blocks per query, and ν associated data blocks in all, and $\tilde{q}$ decryption queries in a NAEAD security game against GOCB$[\Pi, \lambda]$. Then $\forall \mathscr{A} \in \mathbb{A}(q, \tilde{q}, \ell, \sigma)$ we have,* $\mathbf{Adv}^{\mathsf{naead}}_{\mathsf{GOCB}[\Pi,\lambda]}(\mathscr{A}) = \epsilon_{\mathsf{naead}}$, *where*

$$\epsilon_{\mathsf{naead}} \leq \frac{\sigma^2}{2^{n+1}} + \frac{q\sigma + \tilde{q} + 0.5q^2}{2^n - \sigma} + \frac{\tilde{\sigma}(\sigma + 4)}{2^n} + (\sigma + 3\tilde{\sigma})r\epsilon + \tilde{q}\epsilon.$$

Here $\sigma = \mu + \nu$ is the total number of blocks in messages and associated data queried in the encryption queries.

Proof Sketch — The proof of Theorem 1 is given in Section VI. While we do get a factor of $r\sigma\epsilon$ in the bound for GOCB, the original bound of OCB remains intact asymptotically if $r\epsilon = o(2^{\frac{n}{2}})$. To understand why we don't get any degradation in security we have to look into the bound more precisely. Generally the OCB bound is dominated by the probability of collision among the input/output of the underlying block cipher. As per the earlier analyses the collisions can be bounded in terms of the AXU bound of λ, which gives $\sigma^2\epsilon/2$ bound. This evaluates to $\sigma^2 2^{1-n}$ when $\epsilon = 2^{-n}$. However when we replace AXU with LIXU the straightforward analysis will result in security degradation. Instead we first bound the probability of collision between input blocks which are non-local pairs. Since LIXU hashes are perfectly XOR universal on non-local pairs we get a bound of $\sigma^2 2^{1-n}$. Now for each input block there are at most $r - 1$ local pairs and there are at most σ many such input blocks which gives a bound of roughly $r\sigma\epsilon$. Following similar idea we can bound the probability of forgery to $\tilde{q}\sigma 2^{-n} + \tilde{q}r\epsilon$.

C. Instantiating LIXU Hash in GOCB

1) Deriving OCB3 From GOCB *:* We define the mask-generating function used in OCB3 in LIXU setting and derive concrete bound for $n = 128$. OCB3 employs gray code based masking defined using the gray hash function discussed in Section III. As noted earlier gray is a 2^{-128}-AXU hash function.

For OCB3, Krovetz and Rogaway defined the gray function as: For any $i \in \mathbb{N}^+$, $\mathsf{gray}_L(i) = 4\gamma(i) \odot L$. For a fixed $j \in [n]$ (fixed as an application parameter) and $N \in \mathcal{B} \setminus \{0\}$, $\tau(N) = \mathsf{msb}_j(N) \| 0^{n-j}$, and $\beta(N) = \mathsf{lsb}_j(N)$. The OCB3 NAEAD advantage [9] can be obtained from GOCB by instantiating the definition of GOCB as follows:

- $L = (L_1, L_2)$, where L_2 is a keyed function from $\mathcal{B}$ to $\mathcal{B}$, with $L_1 := E_K(0)$ and $\forall N \in \mathcal{B}$, $L_2(N) := \mathsf{sts}_{E_K(\tau(N))}(\beta(N))$.
- For $i \in [2^n]$, and $N \in \mathcal{B} \setminus \{0\}$:
  - $\lambda_1(L, N, i) := L_2(N) \oplus \mathsf{gray}_{L_1}(i)$; $\lambda_5(L, N, i) := \mathsf{gray}_{L_1}(i)$;
  - $\lambda_2(L, N, i) := L_2(N) \oplus \mathsf{gray}_{L_1}(i) \oplus L_1$; $\lambda_6(L, N, i) := \mathsf{gray}_{L_1}(i) \oplus L_1$;
  - $\lambda_3(L, N, i) := L_2(N) \oplus \mathsf{gray}_{L_1}(i) \oplus 2L_1$.
  - $\lambda_4(L, N, i) := L_2(N) \oplus \mathsf{gray}_{L_1}(i) \oplus 3L_1$.

Here we have derived L through K, which adds a insignificant factor in the overall advantage. In [9], all variants of GR were shown to be 2^{-n}-AXU hash over the domain $\{0, 1\}^{128}$. This in combination with Proposition 3, establishes that GR variants are $(2^{-n}, 1)$-LIXU hash. Using Theorem 1 and assuming $\sigma, \tilde{\sigma} < 2^{n-1}$, we get the security bound for the original OCB3 design [9].

Corollary 1. *$\forall \mathscr{A} \in \mathbb{A}(q, \ell, \sigma)$ we have,*

$$\mathbf{Adv}^{\mathsf{naead}}_{\mathsf{OCB3}[E_K^{\pm}, \mathsf{GR}]}(\mathscr{A}) \leq \mathbf{Adv}^{\mathsf{sprp}}_{E_K^{\pm}}(\sigma + \tilde{\sigma} + q + \tilde{q}) + \epsilon_{\mathsf{GR}},$$

where

$$\epsilon_{\mathsf{GR}} = \frac{0.5\sigma^2 + 2q\sigma + q^2 + \tilde{\sigma}\sigma + \sigma + 7\tilde{\sigma} + 3\tilde{q}}{2^n}.$$

2) Instantiating GOCB *With* aes1 *and* aes2 *:* We set $n = 128$. We let $\mathcal{N} \subset \mathcal{B}$, and define

- $\mathsf{A1} : \mathsf{Func}[\mathcal{B}] \times \mathcal{N} \times \mathbb{F}_{2^8} \to \mathcal{B}$ as, $\forall F \in \mathsf{Func}[\mathcal{B}]$, $N \in \mathcal{N}$, $i \in \mathbb{F}_{2^8}$,

$$\mathsf{A1}_F(N, i) := \mathsf{aes1}_{F(N)}(i).$$

- $\mathsf{A2} : \mathsf{Func}[\mathcal{B}] \times \mathcal{B} \times \mathcal{N} \times \mathbb{F}_{2^{32}} \to \mathcal{B}$ as, $\forall (F, L, N, i) \in \mathsf{Func}[\mathcal{B}] \times \mathcal{B} \times \mathcal{N} \times \mathbb{F}_{2^{32}}$,

$$\mathsf{A2}_{(F,L)}(N, i) := \mathsf{aes2}_{(F(N),L)}(i).$$

Next we show that A1 and A2 are LIXU hash functions for appropriate values of ϵ and r. In Lemma 4 and Lemma 5 we show that A1 and A2 are $(2^{-96}, 2^8)$-LIXU and $(2^{-113}, 2^{32})$-LIXU hash functions, respectively. The proofs of these lemmata are given in Section VI.

Lemma 4. *If $\Gamma \leftarrow\!\$\, \mathsf{Func}[\mathcal{B}]$, then $\mathsf{A1}_\Gamma$ is a $(2^{-96}, 2^8)$-LIXU hash function.*

Lemma 5. *If $(\Gamma \times L) \leftarrow\!\$\, \mathsf{Func}[\mathcal{B}] \times \mathcal{B}$, then $\mathsf{A2}_{\Gamma,L}$ is a $(2^{-113}, 2^{32})$-LIXU hash function.*

DESCRIPTION OF GOCB$[E, \mathsf{A1}]$ — We define GOCB based on A1 as follows:

- The maximum message and associated data size per query is limited to 2^6 blocks, i.e. 1 Kilobytes, and the nonce space is $\mathcal{B} \setminus \{0\}$.
- $L = F$, where F is a keyed function from $\mathcal{B}$ to $\mathcal{B}$, where $\forall N \in \mathcal{B}$, $F(N) := E_K(N)$.
- For $i \in [2^6]$, and $N \in \mathcal{B} \setminus \{0\}$:
  - $\lambda_1(F, N, i) := \mathsf{A1}_F(N, 4i + 0)$; and $\lambda_5(F, N, i) := \mathsf{A1}_F(N, 4i + 0) \oplus E_K(0)$.
  - $\lambda_2(F, N, i) := \mathsf{A1}_F(N, 4i + 1)$; and $\lambda_6(F, N, i) := \mathsf{A1}_F(N, 4i + 1) \oplus E_K(0)$.
  - $\lambda_3(F, N, i) := \mathsf{A1}_F(N, 4i + 2)$;
  - $\lambda_4(F, N, i) := \mathsf{A1}_F(N, 4i + 3)$;

  where $4i$ denotes the integer multiplication of i with 4.

Here we need to add a factor of $(q+1)(q+\sigma)2^{-128}$ to bound the probability that some masked input matches with some nonce value. Further a factor of $q^2 2^{-129}$ is required for PRF-PRP switch. Using Theorem 1, and $\sigma, \widetilde{\sigma} < 2^{127}$, we get the security result for GOCB$[E_K, \mathsf{A1}]$ as in Corollary 2.

Corollary 2. *$\forall \mathscr{A} \in \mathbb{A}(q, \tilde{q}, \sigma)$ we have,*

$$\mathbf{Adv}^{\mathsf{naead}}_{\mathsf{GOCB}[E_K^{\pm}, \mathsf{A1}]}(\mathscr{A}) \leq \mathbf{Adv}^{\mathsf{sprp}}_{E_K^{\pm}}(\sigma + q) + \epsilon_{\mathsf{A1}},$$

where

$$\epsilon_{\mathsf{A1}} = \frac{0.5\sigma^2 + 6q\sigma + 2\sigma + \widetilde{\sigma}\sigma + 4\widetilde{\sigma} + 2\tilde{q}}{2^{128}} + \frac{\sigma + 3\widetilde{\sigma}}{2^{88}} + \frac{\widetilde{q}}{2^{96}}.$$

DESCRIPTION OF GOCB$[E, \mathsf{A2}]$ — GOCB$[E, \mathsf{A2}]$ can be defined analogously using mask hash key $L = (L_1, L_2, F)$, where $L_1 = E_K(0)$ and $L_2 = E_K(1)$. We define GOCB based on A2 as follows:

- The maximum message and associated data size per query is limited to 2^{30} blocks, i.e. 16 Gigabytes, and the nonce space is $\mathcal{B} \setminus \{0, 1\}$.
- $L = (L_1, F)$, where $L_1 = E_K(0)$ and F is a keyed function from $\mathcal{B}$ to $\mathcal{B}$. For all $N \in \mathcal{B} \setminus \{0, 1\}$, $F(N) := E_K(N)$.
- For $i \in [2^{30}]$, and $N \in \mathcal{B}$:
  - $\lambda_1(L, N, i) := \mathsf{A2}_L(N, 4i + 0)$; and $\lambda_1(L, N, i) := \mathsf{A2}_L(N, 4i + 0) \oplus E_K(1)$.
  - $\lambda_2(L, N, i) := \mathsf{A2}_L(N, 4i + 1)$; and $\lambda_2(L, N, i) := \mathsf{A2}_L(N, 4i + 1) \oplus E_K(1)$.
  - $\lambda_3(L, N, i) := \mathsf{A2}_L(N, 4i + 2)$;
  - $\lambda_4(L, N, i) := \mathsf{A2}_L(N, 4i + 3)$;

  where $4i$ denotes the integer multiplication of i with 4.

Using similar argument as above we can derive the security bound, given in Corollary 3.

Corollary 3. *$\forall \mathscr{A} \in \mathbb{A}(q, \tilde{q}, \sigma)$ we have,*

$$\mathbf{Adv}^{\mathsf{naead}}_{\mathsf{GOCB}[E_K^{\pm}, \mathsf{A2}]}(\mathscr{A}) \leq \mathbf{Adv}^{\mathsf{sprp}}_{E_K^{\pm}}(\sigma + q) + \epsilon_{\mathsf{A2}},$$

where

$$\epsilon_{\mathsf{A2}} = \frac{0.5\sigma^2 + 6q\sigma + 4\sigma + \widetilde{\sigma}\sigma + 4\widetilde{\sigma} + 2\tilde{q}}{2^{128}} + \frac{\sigma + 3\widetilde{\sigma}}{2^{81}} + \frac{\widetilde{q}}{2^{113}}.$$

3) Extending the Domain of GOCB [E,A1] *and* GOCB [E,A2]*:* The A1 and A2 based instantiations have a restriction on the maximum input length. This is more prominent in case of GOCB$[E, \mathsf{A1}]$, which allows message lengths up to 1 KB. Here we give some ways to extend the domain of GOCB$[E, \mathsf{A1}]$ and GOCB$[E, \mathsf{A2}]$. Our general idea remains the same in all the methods. We improve the maximum input size restriction to roughly 2^{50} bytes or 2^{46} blocks, a usual limit on input size [52]. We need 46 bits to represent each block of input uniquely. We divide the 46-bit length representation into two parts $i \| j$, where j is 6-bit long for GOCB$[E, \mathsf{A1}]$ and j is 30-bit long for GOCB$[E, \mathsf{A2}]$. We will handle the j-value in the same way as before, and employ different techniques to handle (N, i) input of the mask generating function.

1. DOMAIN EXTENSION VIA THE F FUNCTION: We can consider a keyed function over larger domain, say $\mathcal{N} \times [2^{128}]$. For example, consider the definition $F(N, i) := E_K(i \oplus E_K(N))$. This would add an extra $O(\sigma^2/2^n)$ term, a rough upper bound to avoid block cipher input collisions involving $i \oplus E_K(N)$ for some (N, i). This method can clearly extend the message size to 2^{46} but at the cost of an extra block cipher call every time the j value resets to zero, plus it needs one more encryption call for the nonce.
2. DOMAIN EXTENSION VIA NONCE SIZE REDUCTION: Another way is to reduce the nonce size to 88 bits and 112 bits for GOCB$[E, \mathsf{A1}]$ and GOCB$[E, \mathsf{A2}]$, respectively, and let $F(N, i) := E_K(N \| i)$. This is slightly better than the previous method as we can save one block cipher call. As before, this adds an extra $O(\sigma^2/2^n)$ term, a rough upper bound to avoid block cipher input collisions.
3. DOMAIN EXTENSION VIA SUM OF LIXU HASH: We explain the method for GOCB$[E, \mathsf{A2}]$. But similar technique is also applicable to GOCB$[E, \mathsf{A1}]$. We modify the λ function as follows:
   - First, we redefine $L = (L_1, L_2, F, F')$, where $L_1 = E_K(0)$ and $L_2 = E_K(1)$. The nonce space is $\mathcal{N} = \{0, 1\}^{n-1} \setminus \{0, 1\}$, and F and F' are keyed functions from $\mathcal{N}$ to $\mathcal{B}$, defined as $F_K(N) = E_K(N \| 0)$ and $F'_K(N) = E_K(N \| 1)$.
   - Second, $\lambda_1(L, N, i \| j) := \mathsf{A2}_{L_1, F}(N, i) \oplus \mathsf{A2}_{L_2, F}(N, j)$. The other variants are defined analogously as before. Further for AD processing, we XOR $E_K(2)$ (instead of $E_K(1)$) to the modified λ_1 and λ_2 values.

It can be easily shown that this modification offers asymptotically similar security as before.

VI. Security Proofs

In this section we provide the proof for our main result Theorem 1, along with the proofs for Lemma 4 and Lemma 5.

A. Inaptness of the Existing TBC-Based Abstraction

In [8], Rogaway introduced a very nice abstraction for OCB like constructions based on tweakable block ciphers (TBCs). More formally, the TBC abstraction of [8] views the masked input-output block cipher as a way to construct TBCs from block ciphers and efficient mask generating functions, called XEX. In this view any OCB like construction can be viewed as an instance of the TBC-based authenticated encryption called ΘCB, where each encryption TBC call takes a tuple of tweak values (N, i), where N denotes the nonce and i denotes the block index.

This abstraction simplifies as well as modularizes the proof, as it can be shown (using results from [8]) that the NAEAD advantage is bounded by the tweakable strong pseudorandom permutation (TSPRP) advantage of XEX, which is bounded by at most $\sigma^2\epsilon$, where σ denotes the total number of TBC calls and ϵ denotes the AXU bound of the underlying MGF. Consequently, this approach was used in several previous works, most notably [27] and [26].

While the reduction from the NAEAD game for OCB to the TSPRP game for XEX gives tight estimate on the privacy bounds (exactly same as the TSPRP bound for XEX), the reduction is rather loose in case of authenticity bound, as noted in [37]. A straightforward approach gives an authenticity bound of the form $O(\sigma^2\epsilon)+O(1/2^n)$ for single forgery, where σ denotes the number of TBC calls in all encryption queries. The first term here is due to the TSPRP advantage of XEX. Here the $O(\sigma^2\epsilon)$ term is due to the XEX to tweakable random permutation reduction. When extended to multiple verification queries, this results in a bound of the form $O(\tilde{q}\sigma^2\epsilon) + O(\tilde{q}/2^n)$, where $\tilde{q}$ denotes the number of verification queries. Clearly the security degrades for multiple verification queries.

Yet another approach gives a security bound of the form $O(\sigma^2/\epsilon) + O(\tilde{\sigma}^2\epsilon) + O(\tilde{q}/2^n)$, where $\tilde{\sigma}$ denotes the number of TBC calls in all decryption queries. Here the $O(\sigma^2/\epsilon)$ and $O(\tilde{\sigma}^2\epsilon)$ terms are due to the XEX to tweakable random permutation reduction.

A constrained variant of the XEX based abstraction could also be used to get a modularized privacy bound for GOCB. In this abstraction the adversary against tweakable block cipher is restricted to at most r many queries per chunk number. This restriction is perfectly fine, as the NAEAD adversary is nonce-respecting for encryption queries and for each nonce the message length is at most r. This gives a privacy bound of the form $O(\sigma^2/2^n)+O(\sigma r\epsilon)$. But the same is not true for authenticity, as the adversary can make all the decryption queries with tweaks within the same chunk. This results in a term of the form $O(\tilde{\sigma}^2\epsilon)$, which is clearly sub-optimal.

So, we use a more direct approach as also employed in [37]. The proof for Theorem 1 uses coefficient H technique [53] as the main tool for bounding the distinguishing advantages. So, before delving into the proofs we discuss this tool briefly.

B. Coefficient H Technique

We will consider a computationally unbounded and deterministic adversary $\mathscr{A}$ that tries to distinguish $\mathcal{O}_1$ (the real oracle) from $\mathcal{O}_0$ (the ideal oracle) in a security game, in this case NAEAD. We denote the query-response tuple of $\mathscr{A}$'s interaction with its oracle by a transcript ω. Sometime this may also include any additional information the oracle chooses to reveal to the adversary at the end of the query-response phase of the game. We will consider this extended definition of transcript. The probability of realizing a given transcript ω in the security game with an oracle $\mathcal{O}$ is known as the *interpolation probability* of ω with respect to $\mathcal{O}$, denoted $\mathsf{ip}_{\mathcal{O}}[\omega]$. Note that for a transcript to be realized, two things need to happen:

- The adversary needs to make the queries listed in the transcript;
- The oracle needs to make the corresponding responses.

Of these, the former is deterministic; the latter, probabilistic. Thus when we talk of interpolation probability, we are only concerned with the oracle responses, with the assumption that the adversary's queries are consistent with the transcript. For any other adversary, the interpolation probability is trivially 0. Thus $\mathsf{ip}_{\mathcal{O}}[\omega]$ depends only on the oracle $\mathcal{O}$ and the transcript ω and not on the adversary; hence the notation. A transcript ω is said to be *realizable* if $\mathsf{ip}_{\mathcal{O}_0}[\omega] > 0$.

We extend the notation of interpolation probability to a set of transcripts Ω: $\mathsf{ip}_{\mathcal{O}}[\Omega]$ is the probability that the security game with $\mathcal{O}$ results in a transcript $\omega \in \Omega$, and we say that $\mathsf{ip}_{\mathcal{O}}[\Omega]$ is the interpolation probability of Ω with respect to $\mathcal{O}$. Now we state a theorem due to Patarin, known as the Coefficient H Technique [53].

Theorem 2 (Coefficient H Technique [53], [54])**.** *Let Ω be the set of all realizable transcripts. Suppose there is a set $\Omega_{\mathtt{Bad}} \subseteq \Omega$ satisfying the following:*

- $\mathsf{ip}_{\mathcal{O}_0}[\Omega_{\mathtt{Bad}}] \leq \epsilon_{\mathsf{bad}}$;
- *For any $\omega \notin \Omega_{\mathtt{Bad}}$,*

$$\frac{\mathsf{ip}_{\mathcal{O}_1}[\omega]}{\mathsf{ip}_{\mathcal{O}_0}[\omega]} \geq 1 - \epsilon_{\mathsf{ratio}}.$$

Then for an adversary $\mathscr{A}$ trying to distinguish between $\mathcal{O}_1$ and $\mathcal{O}_0$, we have the following bound on its distinguishing advantage:

$$\mathbf{Adv}^{G}_{\mathcal{O}_1}(\mathscr{A}) \leq \epsilon_{\mathsf{bad}} + \epsilon_{\mathsf{ratio}}.$$

Note that we have dropped $\mathcal{O}_0$ from the subscript, as it is completely determined by G. A proof of this theorem is given in multiple papers including [54]–[56]. A weaker version of the result was later rediscovered by Bernstein [57] as the *interpolation theorem*, which was later strengthened by Nandi [58] as the *strong interpolation theorem*.

Additional Notation — For a vector x, $\#x_i$ denotes the multiplicity of x_i in x (number of $j \in |x|$ such that $x_j = x_i$).

We sometime abuse the notation and view x as the set $\{x_i : i \in [|x|]\}$, and denote the cardinality of this set as $\#x$. For $i \in \mathbb{N}^+$, if x is a tuple of i vectors, then x^i denotes the i-th vector in the tuple and for some $j \in [|x^i|]$, x^i_j denotes the j-th coordinate of x^i.

We say a function $f : \mathcal{D} \to \mathcal{R}$ is *partial* or *restricted*, if we know the values of f on a strict subset of $\mathcal{D}$. This subset is called $\mathsf{domain}(f)$. A partial function f can be viewed as a restriction of f to $\mathsf{domain}(f)$. The range of this restricted function is called $\mathsf{range}(f)$. We will treat a partial function as *updatable*: for some $x \in \mathcal{B} \setminus \mathsf{domain}(f)$ and some $y \in \mathcal{R}$, (x, y) may be added to f, so that $\mathsf{domain}(f)$ expands to $\mathsf{domain}(f) \cup \{x\}$, and $\mathsf{range}(f)$ becomes $\mathsf{range}(f) \cup \{y\}$. We say f is *permutation-compatible* if $|\mathsf{domain}(f)| = |\mathsf{range}(f)|$ and f is injective function on the set it is defined.

C. Proof of GOCB NAEAD Bound (Theorem 1)

Let $\mathcal{O}_1^{\pm}$ denote the real oracle corresponding to GOCB, and $\mathcal{O}_0^{\pm}$ denote the ideal oracle corresponding to $(\Gamma, \perp)$. We start off by setting up the necessary notations related to the query-response transcript of $\mathscr{A}$. The transcript generated by $\mathscr{A}$ consists of

- **Encryption query-response tuples:** for $i \in [q]$, the i-th encryption query-response tuple is built of
  - N^i: the queried nonce block, such that for any $i' < i$, $N^i \neq N^{i'}$.
  - A^i: the queried associated data, consisting of a^i complete blocks and an incomplete (possibly empty) block A^i_* at the end;
  - M^i: the queried message, consisting of m^i complete blocks and an incomplete (possibly empty) block M^i_* at the end;
  - C^i: the response ciphertext, such that $|C^i| = |M^i|$;
  - T^i: the response tag block.
- **Decryption query-response tuples:** for $i \in [\widetilde{q}]$ the i-th decryption query-response tuple is built of
  - $\widetilde{N}^i$: the queried nonce block. Note that $\widetilde{N}^i$'s can be repeated.
  - $\widetilde{A}^i$: the queried associated data, consisting of $\widetilde{a}^i$ complete blocks and an incomplete (possibly empty) block $\widetilde{A}^i_*$ at the end;
  - $\widetilde{C}^i$: the queried ciphertext, consisting of $\widetilde{c}^i$ complete blocks and an incomplete (possibly empty) block $\widetilde{C}^i_*$ at the end;
  - $\widetilde{T}^i$: the queried tag block.
  - $\widetilde{M}^i$: the response message, such that $|\widetilde{M}^i| = |\widetilde{C}^i|$. Note that the decryption oracle may return $\perp$, in which case $\widetilde{M}^i$ just denotes the unauthenticated decrypted message to be used internally.

The internal variables arising in one call to the encryption oracle are analogously as given in Algorithm 1 and Figure 1, while the internal variables from the decryption oracle are defined identically, but topped with a tilde to differentiate them from the encryption variables.

Let $\mathcal{I}$ and $\mathcal{J}$ denote the encryption query indices with incomplete-block message and associated data respectively. The decryption counterparts are $\widetilde{\mathcal{I}}$ and $\widetilde{\mathcal{J}}$. Let I and O be multisets defined as

$$I = \{U^i_j : i \in [q],\ j \in [a^i]\} \cup \{U^i_* : i \in \mathcal{J}\} \cup \{X^i_j : i \in [q],\ j \in [m^i] \cup \{\oplus\}\} \cup \{X^i_* : i \in \mathcal{I}\}$$

$$O = \{V^i_j : i \in [q],\ j \in [a^i]\} \cup \{V^i_* : i \in \mathcal{J}\} \cup \{Y^i_j : i \in [q],\ j \in [m^i] \cup \{\oplus\}\} \cup \{Y^i_* : i \in \mathcal{I}\}$$

1) Oracle Behavior and Transcript Extension: Consider a modified security game where we let the real oracle $\mathcal{O}_1$ reveal all the internal input and output blocks appearing in the encryption query phase. Consequently we have to make appropriate behavioral changes in the ideal oracle $\mathcal{O}_0$ to release these internal variables. We describe the sampling behavior of $\mathcal{O}_0$ in greater detail in the subsequent paragraphs. During the sampling $\mathcal{O}_0$ might set $\mathbf{bad} = 1$, whereafter its behavior is undefined.

SAMPLING BEHAVIOR OF $\mathcal{O}_0$

Query Phase: For $i \in [q]$, on the i-th encryption query, for each $j \in [m^i]$, $C^i_j \leftarrow\!\$\, \mathcal{B}$; $T^i \leftarrow\!\$\, \mathcal{B}$; and if $i \in \mathcal{I}$, $\overline{C^i_*} \leftarrow\!\$\, \mathcal{B}$, set $C^i_* = \mathsf{msb}_{|M^i_*|}(\overline{C^i_*})$; return $C^i = C^i_1 \| \cdots \| C^i_{m_i} \| C^i_*$ and T^i to $\mathscr{A}$. For $i \in [\widetilde{q}]$, on the i-th decryption query, return $\perp$ to $\mathscr{A}$.

Post-query Phase: Let $L \leftarrow\!\$\, \mathcal{L}$ and Π, a permutation over $\mathcal{B}$, be undefined on all blocks, i.e $\mathsf{domain}(\Pi) = \mathsf{range}(\Pi) = \emptyset$.
Step 1: Extending the encryption query-response tuple. Set the following values:

- for $i \in [q]$, $j \in [m^i]$ set $X^i_j = M^i_j \oplus \lambda_1(L, N^i, j)$ and $Y^i_j = C^i_j \oplus \lambda_1(L, N^i, j)$;
- for $i \in \mathcal{I}$, set $X^i_* = \lambda_2(L, N^i, m^i)$ and $Y^i_* = \overline{C^i_*} \oplus \mathsf{pad}(M^i_*)$;
- for $i \in [q] \setminus \mathcal{I}$, set $M^i_\oplus = \oplus_{j=1}^{m^i} M^i_j$ and $X^i_\oplus = M^i_\oplus \oplus \lambda_3(L, N^i, m^i)$;
- for $i \in \mathcal{I}$, set $M^i_\oplus = \oplus_{j=1}^{m^i} M^i_j \oplus M^i_*$ and $X^i_\oplus = M^i_\oplus \oplus \lambda_4(L, N^i, m^i)$;
- for $i \in [q]$, $j \in [a^i]$ set $U^i_j = A^i_j \oplus \lambda_5(L, N^i, j)$;
- for $i \in \mathcal{J}$, set $U^i_* = \mathsf{pad}(A^i_*) \oplus \lambda_6(L, N^i, a^i)$;
- set $\mathbf{bad} = 1$, if I or O (the partial multiset containing Y^i_j values for $i \in [q]$ and $j \in [m^i] \cup \{*\}$) contains a non-trivial colliding pair (pair of duplicate elements). Note that all colliding pairs are non-trivial at this stage;
- for all $i \in [q]$, $j \in [m^i]$, fix $\Pi(X^i_j) = Y^i_j$, and for all $i \in \mathcal{I}$, fix $\Pi(X^i_*) = Y^i_*$;
- for all $i \in [q]$, $j \in [a^i]$, fix $\Pi(U^i_j) = V^i_j \leftarrow\!\$\, \mathcal{B} \setminus \mathsf{range}(\Pi)$, and for all $i \in \mathcal{J}$, fix fix $\Pi(U^i_*) = V^i_* \leftarrow\!\$\, \mathcal{B} \setminus \mathsf{range}(\Pi)$;
- for $i \in [q] \setminus \mathcal{J}$, set $S^i = \oplus_{j=1}^{a^i} V^i_j$;
- for $i \in \mathcal{J}$, set $S^i = \oplus_{j=1}^{a^i} V^i_j \oplus V^i_*$;
- for $i \in [q]$ set $Y^i_\oplus = S^i \oplus T^i$.
- set $\mathbf{bad} = 1$, if O (the complete multiset) contains a non-trivial colliding pair (pair of duplicate elements).
- for $i \in [q]$, fix $\Pi(X^i_\oplus) = Y^i_\oplus$.

Note that the partial sampling of Π remains permutation-compatible in step 1 as long as $\mathbf{bad} = 0$.

Step 2: Extending the decryption query-response tuple. Set the following values:

- at the start all intermediate variables are set as undefined.
- for $i \in [\widetilde{q}]$, $j \in [\widetilde{m}^i]$, set $\widetilde{Y}^i_j = \widetilde{C}^i_j \oplus \lambda_1(L, \widetilde{N}^i, j)$; if $i \in \widetilde{\mathcal{I}}$, set $\widetilde{X}^i_* = \lambda_2(L, \widetilde{N}^i, \widetilde{m}^i)$.
- for $i \in [\widetilde{q}]$, $j \in [\widetilde{m}^i]$, if $\widetilde{Y}^i_j \in \mathsf{range}(\Pi)$ then set $\widetilde{X}^i_j = \Pi^-(\widetilde{Y}^i_j)$ and $\widetilde{M}^i_j = \widetilde{X}^i_j \oplus \lambda_1(L, \widetilde{N}^i, j)$; if $i \in \widetilde{\mathcal{I}}$ and $\widetilde{X}^i_* \in \mathsf{domain}(\Pi)$, then set $\widetilde{Y}^i_* = \Pi(\widetilde{X}^i_*)$ and $\widetilde{M}^i_* = \mathsf{msb}_{|\widetilde{C}^i_*|}(\widetilde{Y}^i_*) \oplus \widetilde{C}^i_*$; if $\widetilde{M}^i$ is defined then set $\widetilde{X}^i_\oplus = \widetilde{M}^i_\oplus \oplus \lambda_3(L, \widetilde{N}^i, \widetilde{m}^i)$.
- for $i \in [\widetilde{q}]$, $j \in [\widetilde{a}^i]$, set $\widetilde{U}^i_j = \widetilde{A}^i_j \oplus \lambda_5(L, \widetilde{N}^i, j)$; if $i \in \widetilde{\mathcal{J}}$, set $\widetilde{U}^i_* = \lambda_6(L, \widetilde{N}^i, \widetilde{a}^i)$.
- for $i \in [\widetilde{q}]$, $j \in [\widetilde{a}^i]$, if $\widetilde{U}^i_j \in \mathsf{domain}(\Pi)$ then set $\widetilde{V}^i_j = \Pi(\widetilde{U}^i_j)$; if $\widetilde{V}^i$ is defined then set $\widetilde{Y}^i_\oplus = \widetilde{S}^i \oplus \widetilde{T}^i$.
- At this stage all those intermediate variables, which can be derived through the MGF key L, and adversary's query, are completely determined. Also, some other variables are trivially derived due to the extended encryption transcript.
- set **bad** $= 1$, if there exists $i \in [\widetilde{q}]$, $j \in [\widetilde{m}^i] \cup \{*, \oplus\}$, and one of the following is true:
  - there exists $i' \in [q]$, $j' \in [m^{i'}] \cup \{*, \oplus\}$ such that $(\widetilde{N}^i, \widetilde{C}^i_j) \neq (N^{i'}, C^{i'}_{j'})$ and $\widetilde{Y}^i_j = Y^{i'}_{j'}$.
  - there exists $i' \in [q]$, $j' \in [a^{i'}] \cup \{*\}$ such that $\widetilde{Y}^i_j = V^{i'}_{j'}$.
- set **bad** $= 1$, if there exists $i \in [\widetilde{q}]$, $j \in [\widetilde{a}^i] \cup \{*\}$, and one of the following is true:
  - there exists $i' \in [q]$, $j' \in [m^{i'}] \cup \{*, \oplus\}$ such that $\widetilde{U}^i_j = X^{i'}_{j'}$.
  - there exists $i' \in [q]$, $j' \in [a^{i'}] \cup \{*\}$ such that $(\widetilde{N}^i, \widetilde{A}^i_j) \neq (N^{i'}, \widetilde{A}^{i'}_{j'})$ and $\widetilde{U}^i_j = U^{i'}_{j'}$.

Extend the transcript ω by including all the internal variables computed thus far and return the extended transcript to $\mathscr{A}$.

Identifying the bad transcripts. Let Ω denote the set of all realizable transcripts. We say that $\omega \in \Omega$ is a Bad transcript if it causes **bad** $= 1$. In other words, we say that ω is Bad if one of the following events occur:

EEcoll:
- $\exists\ (i,j) \in [q] \times [m^i] \cup \{*, \oplus\},\ (i', j') \in [q] \times [m^{i'}] \cup \{*, \oplus\}\ :\ (i,j) \neq (i', j') \wedge X^i_j = X^{i'}_{j'}$.
- $\exists\ (i,j) \in [q] \times [a^i] \cup \{*\},\ (i', j') \in [q] \times [a^{i'}] \cup \{*\}\ :\ (i,j) \neq (i', j') \wedge U^i_j = U^{i'}_{j'}$.
- $\exists\ (i,j) \in [q] \times [m^i] \cup \{*, \oplus\},\ (i', j') \in [q] \times [a^{i'}] \cup \{*\}\ :\ X^i_j = U^{i'}_{j'}$.
- $\exists\ (i,j) \in [q] \times [m^i] \cup \{*\},\ (i', j') \in [q] \times [m^{i'}] \cup \{*\}\ :\ (i,j) \neq (i', j') \wedge Y^i_j = Y^{i'}_{j'}$.

ETcoll:
- $\exists\ i \in [q],\ (i', j') \in [q] \times [m^{i'}] \cup \{*\}\ :\ Y^i_\oplus = Y^{i'}_{j'}$.
- $\exists\ i \in [q],\ i' \in [q] \setminus \{i\}\ :\ Y^i_\oplus = Y^{i'}_\oplus$.
- $\exists\ i \in [q],\ (i', j') \in [q] \times [a^{i'}] \cup \{*\}\ :\ Y^i_\oplus = V^{i'}_{j'}$.

DEcoll:
- $\exists\ (i,j) \in [\widetilde{q}] \times [\widetilde{m}^i] \cup \{*\},\ (i', j') \in [q] \times [m^{i'}] \cup \{*, \oplus\}\ :\ (\widetilde{N}^i, \widetilde{C}^i_j) \neq (N^{i'}, C^{i'}_{j'}) \wedge \widetilde{Y}^i_j = Y^{i'}_{j'}$.
- $\exists\ (i,j) \in [\widetilde{q}] \times [\widetilde{m}^i] \cup \{*\},\ (i', j')i' \in [q] \times [a^{i'}] \cup \{*\}\ :\ \widetilde{Y}^i_j = V^{i'}_{j'}$.
- $\exists\ (i,j) \in [\widetilde{q}] \times [\widetilde{a}^i] \cup \{*\},\ (i', j') \in [q] \times [a^{i'}] \cup \{*\}\ :\ (\widetilde{N}^i, \widetilde{A}^i_j) \neq (N^{i'}, A^{i'}_{j'}) \wedge \widetilde{U}^i_j = U^{i'}_{j'}$.
- $\exists\ (i,j) \in [\widetilde{q}] \times [\widetilde{a}^i] \cup \{*\},\ (i', j') \in [q] \times [m^{i'}] \cup \{*, \oplus\}\ :\ \widetilde{U}^i_j = X^{i'}_{j'}$.

DEFcoll:
- $\exists\ i \in [\widetilde{q}],\ i' \in [q]\ :\ (\widetilde{N}^i, \widetilde{A}^i, \widetilde{T}^i) = (N^{i'}, A^{i'}, T^{i'}) \wedge \widetilde{C}^i = C^{i'}_{1..\widetilde{m}^i} \wedge \widetilde{X}^i_\oplus \in I$.

DDFcoll:
- $\exists\ (i,j) \in [\widetilde{q}] \times [\widetilde{m}^i] \cup \{*\},\ i' \in [q]\ :\ (\widetilde{N}^i, \widetilde{A}^i, \widetilde{T}^i) = (N^{i'}, A^{i'}, T^{i'}) \wedge \widetilde{C}^i = C^{i'}_{1..\widetilde{m}^i} \wedge \widetilde{X}^i_\oplus = \widetilde{X}^i_j$.
- $\exists\ (i,j) \in [\widetilde{q}] \times [\widetilde{a}^i] \cup \{*\},\ i' \in [q]\ :\ (\widetilde{N}^i, \widetilde{A}^i, \widetilde{T}^i) = (N^{i'}, A^{i'}, T^{i'}) \wedge \widetilde{C}^i = C^{i'}_{1..\widetilde{m}^i} \wedge \widetilde{X}^i_\oplus = \widetilde{U}^i_j$.
- $\exists\ (i,j) \in [\widetilde{q}] \times [m^i],\ j' \in [\widetilde{m}^i] \setminus \{j\},\ i' \in [q]\ :\ \widetilde{N}^i = N^{i'} \wedge \widetilde{C}^i_j \neq C^{i'}_j \wedge \widetilde{Y}^i_j = \widetilde{Y}^i_{j'}$.
- $\exists\ (i,j) \in [\widetilde{q}] \times [\widetilde{m}^i] \cup \{*, \oplus\}\ :\ \widetilde{Y}^i_1 = \widetilde{Y}^i_j$.

Let $\Omega_{\mathtt{Bad}}$ be the set of all transcripts which are Bad and $\Omega_{\mathtt{Good}} = \Omega \setminus \Omega_{\mathtt{Bad}}$. We bound $\mathsf{ip}_{\mathcal{O}_0}[\Omega_{\mathtt{Bad}}]$ to $O(\sigma^2 2^{-n} + r\sigma\epsilon + q\sigma 2^{-n} + q^2 2^{-n} + \widetilde{\sigma}\sigma 2^{-n} + \widetilde{\sigma} r\epsilon + \widetilde{\sigma} 2^{-n} + \widetilde{q}\epsilon)$ in Lemma 6.

Lemma 6.

$$\mathsf{ip}_{\mathcal{O}_0}[\Omega_{\mathtt{Bad}}] \leq \frac{\sigma^2}{2^{n+1}} + \frac{q\sigma + 0.5q^2}{2^n - \sigma} + \frac{\widetilde{\sigma}(\sigma+4)}{2^n} + (\sigma + 3\widetilde{\sigma})r\epsilon + \widetilde{q}\epsilon.$$

Proof: We bound the probability of getting a bad transcript as follows:

$$\begin{aligned}
&\mathsf{ip}_{\mathcal{O}_0}[\Omega_{\mathtt{Bad}}] \\
&= \Pr[\mathtt{EEcoll} \vee \mathtt{ETcoll} \vee \mathtt{DEcoll} \vee \mathtt{DEFcoll} \vee \mathtt{DDFcoll}] \\
&\leq \Pr[\mathtt{EEcoll}] + \Pr[\mathtt{ETcoll} \mid \neg\mathtt{EEcoll}] \\
&+ \Pr[\mathtt{DEcoll} | \neg(\mathtt{EEcoll} \vee \mathtt{ETcoll})] \\
&+ \Pr[\mathtt{DEFcoll} | \neg(\mathtt{EEcoll} \vee \mathtt{ETcoll} \vee \mathtt{DEcoll})] \\
&+ \Pr[\mathtt{DDFcoll} | \neg(\mathtt{EEcoll} \vee \mathtt{ETcoll} \vee \mathtt{DEcoll} \vee \mathtt{DEFcoll})]
\end{aligned}$$

BOUNDING $\Pr[\mathtt{EEcoll}]$. Let P and Q be a colliding pair. We bound the probability in two cases:

Case 1: The colliding pair is a non-local pair. As λ is a LIXU hash function, we can bound the probability of this case by 2^{-n}. Further we have at most $\sigma(\sigma-1)/2$ many such pairs. Hence

$$\Pr[\mathtt{EEcoll} \ \wedge\ \text{Case 1}] \leq \frac{\sigma^2}{2^{n+1}}.$$

Case 2: The colliding pair is a local pair. This means that the colliding pair must belong to a single message or associated data. We can have two cases: (2.1) The colliding pair belongs to the i-th message; (2.2) The colliding pair belongs to the i-th associated data. Without loss of generality we assume case (2.1). Let $n^i = \lfloor m^i/r \rfloor$ and $r^i = m^i - n^i r$. As λ is an (ϵ, r)-LIXU hash function, we can bound the probability of a fixed pair by ϵ. Hence

$$\begin{aligned}
&\Pr[\mathtt{EEcoll} \ \wedge\ \text{Case 2.1}] \\
&\leq \sum_{i=1}^{q} \sum_{j=1}^{n^i} \binom{r}{2}\epsilon + \sum_{i=1}^{q} \binom{r^i}{2}\epsilon
\end{aligned}$$

$$\overset{[a]}{\leq} r'\epsilon \sum_{i=1}^{q} r \cdot n^i + \sum_{i=1}^{q} \binom{r^i}{2}\epsilon$$
$$\overset{[b]}{\leq} r'\mu\epsilon - \left(r'\epsilon \sum_{i=1}^{q} r^i - \sum_{i=1}^{q} \binom{r^i}{2}\epsilon\right)$$
$$\overset{[c]}{\leq} r'\mu\epsilon.$$

Here $r' = (r-1)/2$; $[a]$ to $[b]$ follows from $rn^i = m^i - r^i$ and $\sum m^i = \mu$; $[b]$ to $[c]$ follows from the fact that $(r-1)r^i \geq r^i(r^i-1)$ for all i. Similarly the probability in case (2.2) can be bounded to $r'v\epsilon$. Hence the total probability in case 2 is bounded by $r'\sigma\epsilon$.

Combining case 1 and 2, we have

$$\Pr[\texttt{EEcoll}] \leq \frac{\sigma^2}{2^{n+1}} + r\sigma\epsilon.$$

BOUNDING $\Pr[\texttt{ETcoll} \mid \neg\texttt{EEcoll}]$. This event bounds the probability that the checksum output block collides with some previous output block given that no other input/output collision occurred. At this instant at most $\sigma = \mu + \nu$ many points are defined for Π, whence we can have at most $q\sigma + q^2/2$ many possible colliding pairs. So we have,

$$\Pr[\texttt{ETcoll} \mid \neg\texttt{EEcoll}] \leq \frac{q\sigma}{2^n - \sigma} + \frac{q^2}{2(2^n-\sigma)}.$$

BOUNDING $\Pr[\texttt{DEcoll}|\neg(\texttt{EEcoll} \vee \texttt{ETcoll})]$. This event bounds the probability that some intermediate output ($\widetilde{Y}^i_j$) or input ($\widetilde{U}^i_j$) input block non-trivially belongs in O or I, respectively. First, suppose $\widetilde{Y}^i_j$ belongs to O for some $(i,j) \in [\widetilde{q}] \times [\widetilde{m}^i]$. Now as in case of EEcoll above: i) the colliding encryption block can either be a non-local pair of (i,j), in which case we bound the probability to at most $\sigma/2^n$; or ii) the colliding encryption block can be a local pair of (i,j), in which case we bound the probability to at most $r\epsilon$. Since there are at most $\widetilde{\mu}$ many decryption ciphertext blocks, the probability that there exists $(i,j) \in [\widetilde{q}] \times [\widetilde{m}^i]$, such that $\widetilde{Y}^i_j$ belongs in O non-trivially is bounded by $\widetilde{\mu}\sigma/2^n + \widetilde{\mu}r\epsilon$. Similarly, one can bound the probability that there exists $(i,j) \in [\widetilde{q}] \times [\widetilde{a}^i]$, such that $\widetilde{U}^i_j$ belongs in I non-trivially to at most $\widetilde{\nu}\sigma/2^n + \widetilde{\nu}r\epsilon$. Combining the two cases, we have

$$\Pr[\texttt{DEcoll}|\neg(\texttt{EEcoll} \vee \texttt{ETcoll})] \leq \frac{\widetilde{\sigma}\sigma}{2^n} + \widetilde{\sigma}r\epsilon.$$

BOUNDING $\Pr[\texttt{DEFcoll}|\neg(\texttt{EEcoll} \vee \texttt{ETcoll} \vee \texttt{DEcoll})]$. This case bounds the probability that some decryption checksum input block collides with some encryption input block with a restriction that the decryption input blocks must all be defined. In this case we have $\widetilde{X}^i_{\oplus} = \sum_{j=1}^{\widetilde{m}^i} \widetilde{M}^i_j \oplus \lambda_3(L, \widetilde{N}, \widetilde{m}^i)$. Now we can have two cases: i) the checksum block collides with $X^{i'}_{\oplus}$, which can be generously bounded by ϵ (assuming $(\widetilde{N}^i, \widetilde{m}^i)$ and $(N^{i'}, m^{i'})$ are local pair); and ii) the checksum block collides with any other $X^{i'}_{j'}$, in which case we bound the probability by 2^{-n} using the 1-universal property of λ. In summary, we have

$$\Pr[\texttt{DEFcoll}|\neg(\texttt{EEcoll} \vee \texttt{ETcoll} \vee \texttt{DEcoll})] \leq \frac{\widetilde{q}}{2^n} + \widetilde{q}\epsilon.$$

BOUNDING $\Pr[\texttt{DDFcoll}|\neg(\texttt{EEcoll} \vee \texttt{ETcoll} \vee \texttt{DEcoll} \vee \texttt{DEFcoll})]$. This case simply bound the probability that there is no collisions within a decryption query. Let us fix a decryption query index $i \in \widetilde{q}$. Then the first case is bounded by at most $(\widetilde{m}^i 2^{-n}$; the second case is bounded by at most $\widetilde{a}^i 2^{-n}$; the third case is bounded by at most $(\widetilde{m}^i - r)2^{-n} + r\epsilon$; and the fourth case is bounded by at most $(\widetilde{m}^i - r)2^{-n} + r\epsilon$. On combining the four cases, we have

$$\Pr[\texttt{DDFcoll}|\neg(\texttt{EEcoll} \vee \texttt{ETcoll} \vee \texttt{DEcoll} \vee \texttt{DEFcoll})] \leq \frac{3\widetilde{\sigma}}{2^n} + 2\widetilde{q}r\epsilon.$$

The result follows by combining all the individual bounds above, and simplifying using $\widetilde{q} \leq \widetilde{\sigma}$. □

Ratio of interpolation probabilities. Fix an arbitrary transcript $\omega \in \Omega_{\texttt{Good}}$. In Lemma 7, we show that the ratio $\mathsf{ip}_{\mathcal{O}_1}[\omega]/\mathsf{ip}_{\mathcal{O}_0}[\omega]$ is at least $(1 - O(\widetilde{q}2^{-n}))$.

Lemma 7. *For any $\omega \in \Omega_{\texttt{Good}}$, we have*

$$\frac{\mathsf{ip}_{\mathcal{O}_1}[\omega]}{\mathsf{ip}_{\mathcal{O}_0}[\omega]} \geq \left(1 - \frac{\widetilde{q}}{2^n - \sigma}\right).$$

Proof: In $\mathcal{O}_0$, for encryption phase: first $\mu + q$ ciphertext and tag outputs are sampled in with replacement fashion, followed by the sampling of ν output blocks in without replacement manner from a subset of $\mathcal{B}$ of size $(2^n - \mu)$ (Π is already defined on μ many input-output blocks); for decryption phase the oracle always returns $\perp$. Hence

$$\mathsf{ip}_{\mathcal{O}_0}[\omega] = \frac{1}{(2^n)^{\mu+q} \cdot (2^n - \mu)^{\underline{\nu}}}. \tag{10}$$

In $\mathcal{O}_1$, for any ω we denote the encryption tuples by ω_e and the decryption tuples by ω_d. Then we have

$$\begin{aligned}\mathsf{ip}_{\mathcal{O}_1}[\omega] &= \Pr_{\mathcal{O}_1}[\omega_e, \omega_d] \\ &= \Pr_{\mathcal{O}_1}[\omega_e] \cdot \Pr_{\mathcal{O}_1}[\omega_d \mid \omega_e] \\ &= \Pr_{\mathcal{O}_1}[\omega_e] \cdot \left(1 - \Pr_{\mathcal{O}_1}[\neg\omega_d \mid \omega_e]\right) \end{aligned} \tag{11}$$

where $\omega_d = (\widetilde{N}^i, \widetilde{A}^i, \widetilde{C}^i, \widetilde{T}^i, \perp)_{i \in [\widetilde{q}]}$, as the ideal oracle always returns $\perp$ on decryption queries. Note that we have slightly abused the notation to use $\neg\omega_d$ as the event that: for some $i \in [\widetilde{q}]$ the i-th decryption query successfully decrypts.

For encryption tuples, exactly $\mu + \nu + q$ many calls are made to Π: one for each of the μ message blocks, one for each of the ν associated data blocks; and one for each of the q tags. As $\sigma = \mu + \nu$ we have

$$\Pr_{\mathcal{O}_1}[\omega_e] = \frac{1}{(2^n)^{\underline{\sigma+q}}}. \tag{12}$$

Now we upper bound the probability of $\neg\omega_d$. It is enough to bound the probability for a fixed index $i \in [\widetilde{q}]$ corresponding to a successful decryption query. By union bound, the probability of $\neg\omega_d$ is at most $\widetilde{q}$ times more than the single forgery probability. Without loss of generality assume that $i \in [\widetilde{q}] \setminus \widetilde{\mathcal{I}}$ and $i \in [\widetilde{q}] \setminus \widetilde{\mathcal{J}}$. Note that the adversary succeeds in forgery

if the adversary somehow makes the i-th query in such a way that the following equation holds.

$$\Pi(\widetilde{X}^i_{\oplus}) \oplus \widetilde{Y}^i_{\oplus} = 0 \tag{13}$$

Now based on the i-th decryption query we can have different cases:

Case 1: $\exists j \in [q]$, $\widetilde{N}^i = N^j$. Based on $\widetilde{C}^i$ we can have two subcases:

Subcase 1.1: $\widetilde{C}^i = C^{i'}_{1..\widetilde{m}^i}$. In this case $\widetilde{M}^i_{\oplus}$ is pre-determined as $\oplus_{k=1}^{\widetilde{c}^i} M^{i'}_k$. Suppose $\widetilde{m}^i = m^{i'}$, i.e. the two ciphertexts are exactly the same. Then we must have some j such that $\widetilde{A}^i_j \neq A^{i'}_j$, otherwise i-th decryption query is a duplicate of i'-th encryption query. Since the transcript is good, $\widetilde{U}^i_j$ is fresh. So, by exploiting the conditional randomness of Π we can bound the probability to at most $1/(2^n - \sigma)$. Now suppose $\widetilde{C}^i$ is a proper prefix of $C^{i'}$, then we must have a fresh $\widetilde{X}^i_{\oplus}$ (as the transcript is good), whence we can again bound the probability to at most $1/(2^n - \sigma)$.

The two cases discussed above are mutually exclusive, hence the probability that the adversary forges in subcase 2.1 is at most $1/(2^n - \sigma)$.

Subcase 1.2: $\widetilde{C}^i \neq C^{i'}_{1..\widetilde{c}^i}$. In this case there exists at least one $k \in [\widetilde{c}^i]$ such that $\widetilde{C}^i_k \neq C^{i'}_k$. We consider first such block $\widetilde{C}^i_k$. Since the transcript is good, we must have a fresh $\widetilde{Y}^i_k$. Observe that we can rewrite Eq. (13) as

$$\Pi^-(\widetilde{Y}^i_k) = \Pi^-(\widetilde{Y}^i_{\oplus}) \oplus \delta \oplus \lambda_3(L, \widetilde{N}^i, \widetilde{m}^i),$$

where δ denotes the checksum of all the blocks of i-th query, except the k-th block. Again, by conditioning on all points except $\widetilde{Y}^i_k$, we bound the probability of this case by $1/(2^n - \sigma)$.

Case 2: $\forall j \in [q]$, $\widetilde{N}^i \neq N^j$. In this case $\widetilde{N}^i$ has not been used for mask generation till now. Without loss of generality, we assume that $\widetilde{m}^i \geq 1$. Since the transcript is good, we must have a fresh $\widetilde{Y}^i_1$. Now using a similar line of argument as in sub case 1.2 we bound the probability of this event by $1/(2^n - \sigma)$.

Note that case 1.1, case 1.2 and case 2 are all mutually exclusive so we can take the maximum over all three. As $\sigma \geq a^i + a^j$ we have

$$\Pr_{\mathcal{O}_1}[\neg\omega_d \mid \omega_e] \leq \sum_{i=1}^{\widetilde{q}} \frac{1}{2^n - \sigma} \leq \frac{\widetilde{q}}{2^n - \sigma} \tag{14}$$

On substituting (12) and (14) in (11) and dividing the result by (10) we have

$$\begin{aligned}\frac{\mathsf{ip}_{\mathcal{O}_1}[\omega]}{\mathsf{ip}_{\mathcal{O}_0}[\omega]} &\geq \frac{(2^n)^{\underline{\mu+q}} \cdot (2^n - \mu)^{\underline{\nu}}}{(2^n)^{\underline{\sigma+q}}} \left(1 - \frac{\widetilde{q}}{2^n - \sigma}\right) \\ &\geq \frac{(2^n)^{\underline{q}}}{(2^n - \sigma)^{\underline{q}}} \left(1 - \frac{\widetilde{q}}{2^n - \sigma}\right) \\ &\geq \left(1 - \frac{\widetilde{q}}{2^n - \sigma}\right).\end{aligned}$$

□

Using Lemma 6, Lemma 7 and Theorem 2 we get the desired result.

D. Proof of Lemma 4 and Lemma 5

The proofs of these lemmata are quite similar. We give the proof for Lemma 4, while the proof for Lemma 5 can be similarly obtained.

Fix distinct $(N, i), (N', i') \in \mathcal{N} \times \mathbb{F}_{2^8}$ and $\delta \in \mathcal{B}$. Now we have two cases:

Case 1: (N, i) and (N', i') are local pairs. We know that $N = N'$ and $i \neq i'$. Hence,

$$\begin{aligned}&\Pr\left[\mathsf{A1}_\Gamma(N, i) \oplus \mathsf{A1}_\Gamma(N', i') = \delta\right] \\ &= \Pr\left[\mathsf{aes1}_{\Gamma(N)}(i) \oplus \mathsf{aes1}_{\Gamma(N)}(i') = \delta\right] \\ &\leq \frac{1}{2^{96}},\end{aligned}$$

where the last inequality follows from Lemma 2.

Case 2: (N, i) and (N', i') are not local pairs. We know that $N \neq N'$. Hence,

$$\begin{aligned}&\Pr\left[\mathsf{A1}_\Gamma(N, i) \oplus \mathsf{A1}_\Gamma(N', i') = \delta\right] \\ &= \Pr\left[\mathsf{aes1}_{\Gamma(N)}(i) \oplus \mathsf{aes1}_{\Gamma(N')}(i') = \delta\right] \\ &\leq \Pr\left[\Gamma(N) = \text{1-AESRD}^{-1}(\mathsf{aes1}_{\Gamma(N')}(i') \oplus \delta) \oplus i\right] \\ &\leq \tfrac{1}{2^{128}}.\end{aligned}$$

The result follows from the bounds in case 1 and case 2.

VII. Software Performance

In this section, we present software implementation of GOCB instantiated with AES-128 as the underlying block cipher, and A1 and A2 as the underlying MGFs, and compare the performance of the proposed designs against the standard OCB3-AES-128. We do the benchmarking in two cases: i) sequential processing (the conventional implementation), and ii) random read access (introduced in this paper).

Implementation — We reuse the optimized C code of OCB3 [9] by Krovetz. Further, we introduce minimal changes, as required, over the OCB3 code to generate the C code for GOCB [AES-128,A1], and GOCB [AES-128,A2]. We also reuse the time measurement mechanism as used in [9]. In summary, we utilized the hardware support on our benchmarking platform for Intel's SSE4 vector instructions [59], Intel's `pclmulqdq` instruction for multiplication over $\mathbb{F}_{2^{128}}$, and Intel's AES-NI library [33] for operations involving AES-128 round functions. We made some simplifying assumptions for the implementations:

- message length is treated to be equal to the length of the associated data, and
- all messages and associated data are assumed to have complete blocks.

Platform — We performed the benchmarking on:

- Intel Core i7-6500U "Skylake" (2.5 GHz with 64 KB L1 cache, 256 KB L2 cache, 4 MB L3 cache) with Ubuntu 16.04 LTS (kernel version 4.4),

which supports Intel's SSE4 vector instructions [59], standard AES-NI instructions [33] and `pclmuldq` [60], the carry-less multiplication instruction.

Setup — The tests for benchmark were compiled using `gcc` version 5.4, with optimization flag `-O3`, and instruction set architecture flag `-march=native` (as instructed in [9]). All

TABLE II
BASELINE THROUGHPUT (CPB) OF AES USING AES-NI

Microarchitecture	Encryption (cpb)
Intel Core i7-6500U "Skylake"	0.66

TABLE III
PERFORMANCE COMPARISON BETWEEN SEQUENTIAL IMPLEMENTATIONS OF OCB3-AES-128, GOCB [AES-128,A1], AND GOCB [AES-128,A2]. THE PERFORMANCE FIGURES PRESENTED ARE THROUGHPUTS, IN UNITS OF CYCLES-PER-BYTE (CPB)

Length	OCB3-AES-128	GOCB[AES-128,A1]	GOCB[AES-128,A2]
128 bytes	1.48	2.11	2.15
256 bytes	1.15	1.41	1.45
512 bytes	0.85	1.07	1.10
1024 bytes	0.75	0.90	0.93
4096 bytes	0.68	0.77	0.81

TABLE IV
SUMMARY OF IPI VALUES FOR OCB3-AES-128, GOCB [AES-128,A1], AND GOCB [AES-128,A2]

	OCB3-AES-128	GOCB[AES-128,A1]	GOCB[AES-128,A2]
IPI:	1.05	1.24	1.28

the tests were run on isolated core, after turning off processor frequency scaling and power management options such as Intel's Turbo Boost and Hyper-Threading technologies. The reference baseline performance for AES-128, using AES-NI instruction set, is presented in Table II. All performance figures presented are throughputs, in units of cycles-per-byte (cpb).

TESTING METHODOLOGY — Following [9], we measure the cpb value for every message length from 1 to 1024 bytes, as well as 1500 and 4096 bytes. To avoid any discrepancies arising due to memory subsystem, we execute warm up runs so that the all code and data is in the cache before benchmarking begins. To capture the average cost of encryption, we use the `rdtsc` instruction — a time-stamp counter available on Skylake processors — to time encryption of the same message: i) 64 times for sequential processing; ii) 1536 times for random access. Note that we average over a larger number of repetitions in random access case to get an appreciable measurement for a single block encryption. The median of 15 such averaged values is reported as the number of cycles. The cycles per byte value is obtained by dividing the median value by the length of the corresponding message.

SEQUENTIAL PROCESSING: RESULTS AND DISCUSSION — Table III summarizes the cpb values for OCB3-AES-128, GOCB [AES-128,A1], and GOCB [AES-128,A2], for moderate to large message lengths, in sequential processing. Following [9], we also present IPI (Internet Performance Index) values — a weighted average cpb vlaue for usual message lengths observed over internet — in Table IV, for

TABLE V
SUMMARY OF CPB VALUES FOR OCB3-AES-128, GOCB [AES-128,A1], AND GOCB [AES-128,A2] TO PROCESS A SINGLE BLOCK OF DATA IN RANDOM ACCESS

OCB3-AES-128	GOCB[AES-128,A1]	GOCB[AES-128,A2]
3.93	3.12	3.29

OCB3-AES-128, GOCB [AES-128,A1], and GOCB [AES-128,A2]. As per our experiments, OCB3 with 10 rounds of AES-128 is approximately, 9% and 10% faster than OCB3 with 11 and 12 rounds, respectively, of AES-128. One may think that GOCB [AES-128,A1] and GOCB [AES-128,A2] should have similar degradation in performance. But the IPI clearly indicates that the A1 and A2 variants are much more slower, approximately 18% and 22%, respectively, than OCB3 over usual internet data. This is mainly due to the lower masking (just after encryption), which is the contentious instruction that breaks the pipelining benefits.

However, the important point to note is that for large messages (i.e. $\geq$ 4096 bytes), which is our main area of focus, the cpb values for GOCB [AES-128,A1] and GOCB [AES-128,A2] tend towards the cpb values for OCB3-AES-128 with 11 and 12 rounds, respectively. Specifically, we have observed that OCB3-AES-128 with 11 and 12 rounds have 0.74 and 0.81 cpbs, respectively, for 4096 bytes message. This is quite close to the cpb values achieved by GOCB [AES-128,A1] and GOCB [AES-128,A2], i.e. 0.77 and 0.81, respectively. Beyond this point the cpb values for GOCB [AES-128,A1] and GOCB [AES-128,A2] are expected to saturate and follow the cpb values of OCB3-AES-128 with 11 and 12 rounds, respectively. In other words, for messages with length $\geq$ 4096 bytes, GOCB [AES-128,A1] and GOCB [AES-128,A2] are only $\approx$ 10% and $\approx$ 20% slower than OCB3.

RANDOM ACCESS: RESULTS AND DISCUSSION— To the best of our knowledge, optimized 128-bit field multiplication algorithms (using `pclmulqdq`) are very close to 5 rounds of AES (using AES-NI) in terms of performance. Thus, theoretically one would expect that GOCB [AES-128,A1], and GOCB [AES-128,A2] will have significantly better random access performance as compared to OCB3, which requires a 128-bit field multiplication.

In Skylake, 1-round AES call requires 4 cycles (latency), so GOCB [AES-128,A1] and GOCB [AES-128,A2] would require close to 44 and 48 cycles, respectively, for AES round calls. Further, roughly 4 cycles are required to create the initial masking state. So, theoretically GOCB [AES-128,A1] and GOCB [AES-128,A2] are expected to have approximately 3-3.1 and 3.2-3.3 cpb. On the other hand, OCB3 requires close to 60 cycles for field multiplication and the AES call. Apart from this approx. 3 cycles are required for gray code generation. So, OCB3 is expected to have around 3.9-4 cpb. In other words, GOCB [AES-128,A1] and GOCB [AES-128,A2] are expected to be close to 25% and 20%, respectively, better than OCB3 in terms of random access. Note that we have ignored the overhead due to nonce processing. This is

reasonable given the assumption that random read/write does not span across different messages.

Table V summarizes the cpb values for OCB3-AES-128, GOCB [AES-128,A1], and GOCB [AES-128,A2], to process an arbitrary single block. Clearly, the experimental data justifies the theoretical speedup described above.

PERFORMANCE IN NON AES-NI ENVIRONMENTS — In environments which lack AES-NI instructions, field multiplication instruction (required for random read in existing OCB designs) is also not available. To the best of our knowledge, optimized implementations of two rounds of AES is expected to be significantly faster than 128 -bit field multiplications. So we expect that A1 and A2 based GOCB will be faster than OCB3, in random access, even on non AES-NI environments, such as GPU based applications.

VIII. CONCLUSION

Most parallel message authentication and authenticated encryption schemes use certain masking routines based on the position of a message block and/or a nonce. We explicitly define this idea as the *Mask Generating Function (MGF)*. We used this abstraction of the masking scheme to explore efficient solutions for random read access in OCB variants. Specifically, the MGF based generalization of OCB design, called GOCB, allows us to prove that a relaxed notion of universal hash — Locally-Imperfect XOR Universal Hash (LIXU) — is sufficient to provide adequate security guarantees for masking. We demonstrate secure instantiations of LIXU hash functions in GOCB design using 1-round and 2-round AES, which are expected to make direct computation extremely efficient. This in turn allows for efficient random read access in GOCB. We demonstrate that even though 1-round and 2-round AES do not have enough differential probability (MDP) to qualify as optimal AXU hash functions, they do just fine as MGFs in the GOCB design if we consider the LIXU security notion for masking. The abstraction of LIXU hash functions for mask generation, and the proposed instantiation of LIXU hash using 1-round and 2-round AES in GOCB like construction are expected to have a number of benefits in practice, including — random read access, out-of-sequence encryption/decryption, high parallelism, and support for constant time implementations — to name a few. It will be interesting to see tailor-made applications and instantiations of LIXU hash functions in other practical cryptographic constructions in the future. For instance, it would be interesting to see whether the XTS mode by Rogaway [8] can be modified using LIXU hash function.

REFERENCES

[1] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and Analysis of the generic composition paradigm," *J. Cryptol.*, vol. 21, no. 4, pp. 469–491, Oct. 2008.

[2] C. Jutla, "Encryption modes with almost free message integrity," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.*, Apr. 2001, pp. 529–544.

[3] P. Rogaway, M. Bellare, and J. Black, "OCB: A block-cipher mode of operation for efficient authenticated encryption," *ACM Trans. Inf. Syst. Secur.*, vol. 6, no. 3, pp. 365–403, Aug. 2003.

[4] F. Abed, C. Forler, and S. Lucks, "General classification of the authenticated encryption schemes for the CAESAR competition," *Comput. Sci. Rev.*, vol. 22, pp. 13–26, Nov. 2014.

[5] A. Chakraborti, A. Chattopadhyay, M. Hassan, and M. Nandi, "TriviA: A fast and secure authenticated encryption scheme," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.*, Sep. 2015, pp. 330–353.

[6] P. Sarkar, "Modes of operations for encryption and authentication using stream ciphers supporting an initialisation vector," *Cryptogr. Commun.*, vol. 6, no. 3, pp. 189–231, Sep. 2014.

[7] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "Duplexing the sponge: Single-pass authenticated encryption and other applications," in *Proc. Int. Workshop Sel. Areas Cryptogr.*, 2011, pp. 320–337.

[8] P. Rogaway, "Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2004, pp. 16–31.

[9] T. Krovetz and P. Rogaway, "The software performance of authenticated-encryption modes," in *Proc. Int. Workshop Fast Softw. Encryption*, 2011, pp. 306–327.

[10] D. A. McGrew and J. Viega, "The security and performance of the galois/counter mode (GCM) of operation," in *Proc. Int. Conf. Cryptol. India*, 2004, pp. 343–355.

[11] E. Andreeva, A. Bogdanov, A. Luykx, B. Mennink, E. Tischhauser, and K. Yasuda, "Parallelizable and authenticated online ciphers," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2013, pp. 424–443.

[12] L. Bossuet, N. Datta, C. Mancillas-López, and M. Nandi, "ELmD: A pipelineable authenticated encryption and its hardware implementation," *IEEE Trans. Comput.*, vol. 65, no. 11, pp. 3318–3331, Nov. 2016.

[13] N. Datta, A. Luykx, B. Mennink, and M. Nandi, "Understanding RUP integrity of COLM," *IACR Trans. Symmetric Cryptol.*, vol. 2, pp. 143–161, Aug. 2017.

[14] J. Jean, I. Nikolić, and T. Peyrin, "Tweaks and keys for block ciphers: The TWEAKEY framework," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2014, pp. 274–288.

[15] T. Peyrin and Y. Seurin, "Counter-in-tweak: Authenticated encryption modes for tweakable block ciphers," in *Proc. Annu. Int. Cryptol. Conf.*, 2016, pp. 33–63.

[16] M. Dworkin, "Recommendation for block cipher modes of operation—Methods and techniques," U.S. Dept. Commerce, Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. NIST SP 800-38A, 2001.

[17] D. Whiting, R. Housley, and N. Ferguson, *Counter with CBC-MAC (CCM)*, document RFC-3610, Internet Eng. Task Force, 2003.

[18] P. Rogaway. *Authenticated-Encryption Software Performance: Comparison of CCM, GCM, and OCB*. Accessed: Jun. 19, 2018. [Online]. Available: http://web.cs.ucdavis.edu/ rogaway/ocb/performance/

[19] *How Much Faster is OCB Than CCM or GCM?*. Accessed: Jun. 19, 2018. [Online]. Available: http://web.cs.ucdavis.edu/ rogaway/ocb/ocb-faq.htm#performance

[20] J. Postel, *User Datagram Protocol*, document RFC-768, Internet Eng. Task Force, 1980.

[21] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman, *The Secure Real-Time Transport Protocol (SRTP)*, document RFC-3711, 2004.

[22] I. Johansson and M. Westerlund, *Support for Reduced-Size Real-Time Transport Control Protocol (RTCP): Opportunities and Consequences*, document RFC-5506, 2009.

[23] J. Lennox, *Encryption of Header Extensions in the Secure Real-Time Transport Protocol (SRTP)*, document RFC-6904, 2013.

[24] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, document RFC-3550, 2003.

[25] J. Black and P. Rogaway, "A block-cipher mode of operation for parallelizable message authentication," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.*, Amsterdam, The Netherlands, Apr./May 2002, pp. 384–397.

[26] R. Granger, P. Jovanovic, B. Mennink, and S. Neves, "Improved masking for tweakable blockciphers with applications to authenticated encryption," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2016, pp. 263–293.

[27] D. Chakraborty and P. Sarkar, "A general construction of tweakable block ciphers and different modes of operations," *IEEE Trans. Inf. Theory*, vol. 54, no. 5, pp. 1991–2006, May 2008.

[28] S. Contini and Y. Yin, "On differential properties of data-dependent rotations and their use in MARS and RC6," in *Proc. 2nd AES Conf.*, 2000, pp. 230–239.

[29] K. Minematsu, "A short universal hash Function from bit rotation, and applications to blockcipher modes," in *Proc. Int. Conf. Provable Secur.*, 2013, pp. 221–238.
[30] S. Park, S. H. Sung, S. Lee, and J. Lim, "Improving the upper bound on the maximum differential and the maximum linear hull probability for SPN structures and AES," in *Proc. Int. Workshop Fast Softw. Encryption*, 2003, pp. 247–260.
[31] L. Keliher, "Refined analysis of bounds related to linear and differential cryptanalysis for the AES," in *Proc. Int. Conf. Adv. Encryption Standard*, 2004, pp. 42–57.
[32] L. Keliher and J. Sui, "Exact maximum expected differential and linear probability for two-round advanced encryption standard," *IET Inf. Secur.*, vol. 1, no. 2, pp. 53–57, Jun. 2007.
[33] S. Gueron, "Intel Advanced Encryption Standard (AES) new instructions set," Intel Corporation, Santa Clara, CA, USA, White Papers, 2010. [Online]. Available: https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf
[34] K. Minematsu, "Improved security analysis of XEX and LRW modes," in *Proc. Int. Workshop Sel. Areas Cryptogr.*, 2006, pp. 96–113.
[35] K. Minematsu and Y. Tsunoo, "Provably secure MACs from differentially-uniform permutations and AES-based implementations," in *Proc. Int. Workshop Fast Softw. Encryption*, 2006, pp. 226–241.
[36] M. Bellare, V. T. Hoang, and S. Keelveedhi, "Cryptography from compression functions: The UCE bridge to the ROM," in *Proc. Annu. Cryptol. Conf.*, 2014, pp. 169–187.
[37] R. Bhaumik and M. Nandi, "Improved security for OCB3," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, Nov. 2017, pp. 638–666.
[38] H. Krawczyk, "LFSR-based hashing and authentication," in *Proc. Annu. Int. Cryptol. Conf.*, 1994, pp. 129–139.
[39] P. Rogaway, "Bucket hashing and its application to fast message authentication," *J. Cryptol.*, vol. 12, no. 2, pp. 91–115, Mar. 1999.
[40] K. Minematsu, "Parallelizable rate-1 authenticated encryption from pseudorandom functions," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2014, pp. 275–292.
[41] K. Yasuda, "On the full MAC security of a double-piped mode of operation," *IEICE Trans.*, vol. 94, no. 1, pp. 84–91, Jan. 2011.
[42] S. Halevi and P. Rogaway, "A tweakable enciphering mode," in *Proc. Annu. Int. Cryptol. Conf.*, 2003, pp. 482–499.
[43] T. Krovetz and P. Rogaway, *The OCB Authenticated-Encryption Algorithm*, document RFC-7253, 2014.
[44] M. Ågren, M. Hell, T. Johansson, and W. Meier, "Grain-128 a: A new version of Grain-128 with optional authentication," *Int. J. Wireless Mobile Comput.*, vol. 5, no. 1, pp. 48–59, 2011.
[45] J. Daemen and V. Rijmen, *The Design Rijndael: AES—The Advanced Encryption Standard*. New York, NY, USA: Springer, 2002.
[46] J. Daemen and V. Rijmen, "Understanding two-round differentials in AES," in *Proc. Int. Conf. Secur. Cryptogr. Netw.*, 2006, pp. 78–94.
[47] "Announcing the advanced encryption standard (AES)," U.S. Dept. Commerce, Federal Inf. Process. Standards, Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. Publication FIPS 197, 2001. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf
[48] M. Matsui and A. Yamagishi, "A new method for known plaintext attack of FEAL cipher," in *Proc. Workshop Theory Appl. Cryptograph. Techn.*, 1992, pp. 81–91.
[49] E. Biham and A. Shamir, "Differential cryptanalysis of DES-like cryptosystems," *J. Cryptol.*, vol. 4, no. 1, pp. 3–72, Jan. 1990.
[50] G. Jakimoski and K. P. Subbalakshmi, "On efficient message authentication via block cipher design techniques," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2007, pp. 232–248.
[51] D. J. Bernstein. (2005). *Cache-Timing Attacks on AES*. [Online]. Available: https://cr.yp.to/antiforgery/cachetiming-20050414.pdf
[52] N. L. S. Team. *Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process*. Accessed: Apr. 09, 2019. [Online]. Available: http://web.cs.ucdavis.edu/ rogaway/ocb/ocb-faq.htm#performance
[53] J. Patarin, "Etude des générateurs de permutations pseudo-aléatoires basés sur le schéma du DES," Ph.D. dissertation, Université de Paris, Paris, France, 1991.
[54] J. Patarin, "The 'Coefficients H' Technique," in *Proc. Int. Workshop Sel. Areas Cryptogr.*, 2008, pp. 328–345.
[55] B. Mennink and S. Neves, "Encrypted davies-meyer and its dual: Towards optimal security using mirror theory," in *Proc. Annu. Int. Cryptol. Conf.*, 2017, pp. 556–583.
[56] S. Chen and J. Steinberger, "Tight security bounds for key-alternating ciphers," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2014, pp. 327–350.
[57] D. J. Bernstein, "How to stretch random functions: The security of protected counter sums," *J. Cryptol.*, vol. 12, no. 3, pp. 185–192, 1999.
[58] M. Nandi, "A simple and unified method of proving indistinguishability," in *Proc. Int. Conf. Cryptol. India*, 2006, pp. 317–334.
[59] *SSE4 (AES) Programming Reference*, Intel Corporation, Santa Clara, CA, USA, 2007.
[60] S. Gueron, "Intel carry-less multiplication instruction and its usage for computing the GCM mode," Intel Corporation, Santa Clara, CA, USA, White Papers, 2011. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/carry-less-multiplication-instruction-in-gcm-mode-paper.pdf

Ashwin Jha received his B.E. degree in computer engineering from the Delhi College of Engineering, Delhi, India in 2012, and M.Tech. degree in computer science from the Indian Statistical Institute, Kolkata, India in 2015. Currently, he is a Ph.D. student at the Indian Statistical Institute, Kolkata, India. His research interests include cryptology and pseudorandomness.

Cuauhtemoc Mancillas-López received his B.E. degree in electronic and communications engineering from ESIME-IPN, Mexico, in 2004, and M.Sc. and Ph.D. degree in computer science from CINVESTAV-IPN, Mexico, in 2007 and 2013 respectively. He did a post-doc in the Hubert Curien Laboratory at the University of Lyon at Saint Etienne, France. Currently, he is a researcher in the Department of Computer Science, CINVESTAV-IPN, Mexico City. His current research interests include design and analysis of provably secure symmetric encryption schemes, efficient software/hardware implementations of cryptographic primitives, and computational arithmetic.

Mridul Nandi received his B.Stat., M.Stat., and Ph.D. degree in computer science from the Indian Statistical Institute, Kolkata, India in 1999, 2001, and 2005, respectively. Currently he is an associate professor at the Applied Statistics Unit, Indian Statistical Institute, Kolkata, India. His research interests include design and analysis of symmetric-key schemes; efficient software/hardware implementations of cryptographic algorithms; functional encryption schemes and signature schemes.

Sourav Sen Gupta received his B.E. degree in electronics and telecommunication engineering from Jadavpur University, India, in 2006, M.Math. degree in pure mathematics from University of Waterloo, Canada, in 2008, and Ph.D. degree in computer science from Indian Statistical Institute in 2014.

From 2014 to 2017, he was a Lecturer cum Post-Doctoral Fellow with the R C Bose Centre for Cryptology and Security, Indian Statistical Institute, Kolkata. Since 2018, he has been a Lecturer with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. His research interests include cryptanalysis, cryptographic design, cryptocurrencies, blockchain technology, and applications of machine learning in cybersecurity.