A New Authenticated Encryption Technique for Handling Long Ciphertexts in Memory Constrained Devices

Megha Agrawal, Donghoon Chang, and Somitra Sanadhya

Indraprastha Institute of Information Technology, Delhi (IIIT-D), India
{meghaa,donghoon,somitra}@iiitd.ac.in

Abstract. In authenticated encryption schemes, there are two techniques for handling long ciphertexts while working within the constraints of a low buffer size: Releasing unverified plaintext (RUP) or Producing intermediate tags (PIT). In this paper, in addition to the two techniques, we propose another way to handle a long ciphertext with a low buffer size by storing and releasing only one (generally, or only few) intermediate state without releasing or storing any part of an unverified plaintext and without need of generating any intermediate tag.
In this paper we explain this generalized technique using our new construction sp-AELM. sp-AELM is a sponge based authenticated encryption scheme that provides support for limited memory devices. We also provide its security proof for privacy and authenticity in an ideal permutation model, using a code based game playing framework. Furthermore, we also present two more variants of sp-AELM that serve the same purpose and are more efficient than sp-AELM.
The ongoing CAESAR competition has 9 submissions which are based on the Sponge construction. We apply our generalized technique of storing single intermediate state to all these submissions, to determine their suitability with a Crypto module having limited memory. Our findings show that only ASCON and one of the PRIMATE's mode(namely GIBBON) satisify the limited memory constraint using this technique, while the remaining schemes (namely, Artemia, ICEPOLE, Ketje, Keyak, NORX, Π-cipher, STRIBOB and two of the PRIMATEs mode: APE & HANUMAN) are not suitable for this scenario directly.

Keywords: Authenticated encryption, CAESAR, Cryptographic module, Remote key authenticated encryption, Decrypt-then-mask protocol, Privacy, Authenticity.

1 Introduction

Authenticated encryption (AE), formalized in [3, 4], is a technique that combines encryption and authentication to provide both privacy and authenticity of the data by means of a single construction, usually with a single key. The formal definition of privacy and authenticity are provided in [4], where AE is introduced as a block cipher mode of operation with the same block cipher performing both encryption and authentication. Informally, an AE scheme receives a message and a secret key as inputs and generates a ciphertext and a tag during encryption process. On the receiver side, the ciphertext and tag pair is verified, and the corresponding plaintext after decryption is returned only if the AE scheme verifies the supplied tag.

There exist a wide variety of AE modes. Some of the popular AE modes are GCM [28], OCB [30], EAX [6], CCM [15] and CWC [24]. In these "offline" AE modes, the device needs to store the complete plaintext, in the encryption mode, to produce the tag. Not all devices possess large memory to completely store a message and hence the modes mentioned earlier can't be directly used in the case of a long message. Consequently, the concept of online authenticated encryption was proposed in [18], in which encryption can be done on the fly. If the message (resp. ciphertext) blocks are denoted by $M_1, M_2, \ldots$ (resp. $C_1, C_2, \ldots$), then the requirement of online AE means that the ciphertext block C_i can be computed without the knowledge of plaintext block M_j for any $j > i$. Most of the block-cipher based authenticated encryption schemes can be used in online encryption mode. Many authenticated encryption schemes that support online encryption have been submitted to the currently on going CAESAR competition [1]. Some of these are APE [2], NORX [23], AEGIS [31] etc.

However, decryption in an AE scheme can never be online due to the fact that the ciphertext needs to be verified before producing the plaintext. If this was not the case then an attacker could simply ask for the decryption of a ciphertext of his choice, while associating any arbitrary tag with the ciphertext. The requirement of verifying the tag before producing the decrypted text can be solved in two different ways. In one case, the system implementing the decryption process could store the complete plaintext and produce it only if the tag verifies. In the other case, the system first applies the tag verification algorithm and then produces the plaintext, block by block, only if the tag verifies. It may be possible to achieve the implementation of the first method in a single pass over the ciphertext, whereas the second method can't be implemented in less than 2 passes over the ciphertext. However, the first method requires potentially large memory on the device while the second method could be implemented even on low memory devices. In fact, with the increasing usage of low cost RFID devices, sensors and trusted platform modules (TPM), the need for an *AE* scheme which can supports both the encryption and the decryption functions in online form is increasing day by day. None of the schemes submitted in the CAESAR competition consider this limited memory constraint explicitly.

Related work: At FSE 1996, Blaze [10] proposed a new paradigm for secret-key block ciphers: Remotely keyed encryption (RKE). RKE is concerned with the problem of "high-bandwidth encryption with low bandwidth smartcards". A scheme to achieve the same was proposed in this work, but some limitations of the scheme were also mentioned in the same work. Later, Lucks [25] provided the first formal model for RKE and chose to interpret the question as that of implementing a remotely key pseudorandom permutation (RKPRP). The work [25] was further improved, both in terms of formal modelling and the actual construction, by an influential work of Blaze et. al [11]. It was observed in this later work that the PRP's length preserving property implies that it can not be semantically secure when viewed as encryption function alone. Thus, in addition to the RKPRP which was termed "length preserving RKE", the work [11] also introduced the notion of "length increasing RKE" which was also referred to as "remote key authenticated encryption" (RKAE). While the definition given in [11] was important and the first step towards formalizing this new notion, it turns out to be quite non-standard (it involves an "arbiter" who can fool any adversary). The notion of such an arbiter looks quite artificial. Later, Dodis et al. [14] provided the formal definition of a remote key authenticated encryption. The RKAE scheme solves the problem where one wishes to split the task of high bandwidth authenticated encryption between a secure client (but limited bandwidth or computationally limited device) and an insecure (but computationally powerful) host. Though RKAE *is efficient, but the Crypto device (e.g. a card) that performs encryption and decryption doesn't know the actual value of plaintext and ciphertext* (Refer Table 1). *Rather, it trusts the insecure host to provide these values, making it unsuitable for real world applications.* Further, the security of this scheme is proved in a setting where the adversary has oracle access only to the combined functionality of the host and the card. However, the host is assumed to be insecure (subject to break-in by an adversary) in this scheme, i.e., the adversary can have access to the internal state of an encryption algorithm executed on the host side. So, the underlying assumption of having oracle access to combined functionality is not justified for defining security of the scheme.

Later, Fouque et al. at SAC 2003 proposed a decryption protocol named Decrypt-Then-Mask (DTM) [18]. The main idea behind this protocol is to blind the plaintext blocks obtained after decryption by XORing them with a pseudorandom sequence of bits and return it to the sender. These blocks do not "look" meaningful to the attacker until "unmasking" is applied. Once a tag gets verified at the receiver end, the seed of the pseudorandom number generator(PRNG) used to mask the plaintext blocks is returned to the sender. This allows the sender to run the same PRNG and un-mask the plaintext blocks. However this protocol has a drawback. *The DTM requires two additional passes excluding no. of passes required for underlying AE during decryption, which makes it computationally expensive, specially in the case of long messages.* These two extra passes (one at the crypto module and another one at the user side) in DTM are due to the use of PRNG, which creates extra overhead particularly in case of long message. The scenarios mentioned above point towards the need of an authenticated encryption scheme which remains efficiently implementable with low memory crypto module even while handling long messages.

Recently, Andreeva et. al in ASIACRYPT 2014 [16] provides definitions to formalize an AE scheme's security against release of unverified plaintexts. Even though these settings provide support for the devices with limited memory constraint but from the security point of view, it is essential to have a tag verification before releasing the plaintext.

The issue of releasing unverified plaintest has also been discussed in on going CAESAR competition [1]. Its feature page explicitly states: *"Beware that security questions are raised by any authenticated cipher that handles a long ciphertext in one pass without using a large buffer: releasing unverified plaintext to applications often means releasing it to attackers and also requires an analysis of how the applications will react"*. However releasing unverified plaintext is not the only solution to handle large ciphertexts in memory constrained environment. In this work, we present a new solution that serves the same purpose, even though, it requires two passes, among them only one pass is effective as another pass takes place on user side and we usually assume that the user has enough memory on their side.

Another requirement regarding intermediate tags mentioned in CAESAR [1] feature page states: *"If a long plaintext is split into separate packets, each of which is separately authenticated (and encrypted), then a long forgery need not be buffered before it is rejected. Applying this split to any MAC (or authenticated cipher) produces* *a new MAC (or authenticated cipher) with different performance properties; perhaps the same type of fast rejection can be better achieved in another way*". These settings still require significant memory to store the decrypted text and tags for every packet, which may not be available for the devices with the limited memory constraint. However the scheme proposed in this paper can support intermediate tag even for devices with the limited memory as we only need to store one intermediate state for all the packets and can also support fast rejection by checking those intermediate tags. For more details one can refer to Appendix A.

Our Contribution: In this paper, we introduced a new generalized technique for authenticated encryption which store only one intermediate state instead of entire plaintext during tag verification. And then this intermediate state is used by the user on their side to decrypt ciphertext. Without releasing unverified plaintext this technique allow us to do a tag verification for a long message on the devices with limited memory settings. We explained this generalized technique using our new construction sp-AELM and its variants. This new construction and its two more variants support both online encryption as well as tag verification in constrained memory setting for long messages, which makes it suitable for implementation on devices that have limited memory. The Sponge construction supports arbitrarily long input and output sizes, and hence allows building various cryptographic primitives such as a hash function or a stream cipher [8]. More recently, use of Sponge based hash functions as an authenticated encryption primitive has been proposed in [7]. Keccak, the winner of the SHA-3 competition, is also built on the Sponge construction. Considering the acceptability and future adaptability of the Sponge construnction in Cryptographic software/hardware, we choose the Sponge function as a basic primitive for the construction of our authenticated encryption scheme sp-ALEM.
In sp-AELM, instead of returning all the plaintext blocks, we provide only one internal state to the user, using which plaintext can be computed easily at his side. Our scheme requires a total of three passes, one for encryption and two for decryption. This compares favourably with the DTM scheme, which requires overall more number of passes (discussed in Table 1), making our scheme more efficient. In sp-AELM, all computations are proposed to be done inside a cryptographic module, where the actual plaintext and ciphertext values are securely known, as opposed to the RKAE scheme [14]. More details about sp-AELM are given in Section 4. In addition to this, we also present two more efficient variants of sp-AELM. More detail about these variants are given in Section 6.

Table 1 Description: This table shows the comparison of RKAE, DTM, the proposed scheme sp-AELM and sp-AELM variants. Our proposed scheme and its variants support online encryption. However, in the case of RKAE and DTM, it depends on the underlying AE scheme that is used. All these three schemes support low memory verification, i.e., there is no need to store message blocks on a cryptographic module during decryption. Next we compare these schemes on the basis of number of passes required for encryption and decryption. For a given message M of n blocks, we use the usual notion of a "pass" as the processing of all n blocks of M once. The RKAE scheme requires two passes[1] for encryption during execution of Conceal algorithm (one pass for encrypting n block message and another pass for computing the hash value on the output of the first pass, explained in Fig. 1(a)), while the number of passes required for DTM depends on the underlying AE scheme. sp-AELM and its variants require only one pass for encryption i.e. it visits the n block message exactly once. For decryption and verification, RKAE requires two passes[1] as shown in Fig. 1(b) (one pass for computing the hash value and another one for decryption). DTM uses two additional passes for pseudo random number generator (one pass for masking the n block message and another for unmasking the n blocks) apart from the number of passes for the underlying authenticated decryption algorithm. sp-AELM and its variants require only two passes (one at the receiver side and another one at the sender side as shown in Fig. 4(b)) for the decryption and verification. We have also used communication overhead between the host and the Crypto module as a comparison parameter in terms of the number of message blocks n. RKAE require only 2 communications(shown in Fig. 1b in), DTM require $2n + 2$ communications(shown in Fig. 2(b)) whereas sp-AELM and its variants require $n + 2$ communications(shown in Fig. 4(b)).

[1] For RKAE encryption/decryption we neglect the computations carried out at the Crypto module, as it is processed on a small data.

Parameters	**RKAE** [14]	**DTM** [18]	**sp-AELM** [This paper]	**sp-AELM variants** [This paper]
Online Encryption	Depends on underlying AE	Depends on underlying AE	Yes	Yes
Support for low memory verification	Yes	Yes	Yes	Yes
No. of Passes required for Encryption and tag generation	Two	Depends on underlying AE	One	One
No. of Passes required for Decryption and Verification	Two	2 + Depends on underlying AE	Two	Two
Communications overhead b/w Host and Crypto module during verification and decryption	2	$2n+2$	$n+2$	$n+2$
Knowledge of Plaintext and Ciphertext to Crypto module	No	Yes	Yes	Yes

Table 1. Comparison of RKAE, DTM, sp-AELM and sp-AELM variants:(a) One pass is defined as processing of n blocks of a message once.(b)The communication overhead is given in terms of number of blocks n for a message M.

In addition to this, we also applied the same technique used in sp-AELM on all Sponge based authenticated encryption schemes submitted to the CAESAR competition, to evaluate their suitability for supporting low memory constraint. Currently there are nine Sponge based AE schemes in the CAESAR competition. We addressed all these nine schemes using the same technique that we used in sp-AELM, i.e., during verification and decryption, storing only one intermediate state instead of storing all decrypted text and returning this intermediate state only when tag gets verified. Detailed analysis of the individual scheme is provided in Section 7. Our findings are briefly summarized in Table 2.

Sponge based AE Schemes submitted in CAESAR	**Support for limited memory devices** (shown in Section 7)
Artemia [22], ICEPOLE [29], Ketje [20], Keyak [21], NORX [23], PRIMATEs(APE, HANUMAN) [17], π-Cipher [13], STRIBOB [27]	No
Ascon [12], PRIMATEs(GIBBON) [17]	Yes

Table 2. Analysis of CAESAR schemes showing their suitability for supporting limited memory constraint.

Roadmap : The rest of this paper is organized as follows: Section 2 gives the brief description of Remote Key Authenticated Encryption [14] and the protocol defined in [18]. Section 3 defines some preliminaries followed by Section 4 that defines the construction of the proposed AE scheme (sp-AELM). We provide the security proof of the proposed scheme in Section 5. In Section 6, we present two more efficient variants of sp-AELM. Section 7 presents the brief analysis of all Sponge base AE schemes submitted to the CAESAR competition. Finally, we conclude this paper with Section 8.

2 Previous Work

2.1 Remotely Keyed Authenticated Encryption

In [14] Dodis et al. proposed a formalized solution to the problem of remotely keyed authenticated encryption (RKAE). One round of the proposed scheme consists of seven different algorithms that run between two parties called the host and the cryptographic module: (RKG, StartAE, CardAE, FinishAE, StartAD, CardAD, FinishAD). In this scheme, the authors used the smart card as the Crypto module. The host is assumed to be powerful but insecure, and the Crypto module is assumed to have low bandwidth and limited memory but taken to be secure. For a given authenticated encryption scheme AE, we explain the RKAE using an example. First, the RKG runs and produces the secret key K and this key gets stored on the Crypto module. The process of encryption is divided into 3 parts (shown in Fig. 1(a)):

1. First, the host runs a probabilistic algorithm *StartAE* on input M (which we conveniently rename *Conceal* and produces (h, b) pair, where $|b| << |h|$, $h \leftarrow E_\tau(M), b = \tau \| H(h)$, E is any one time symmetric encryption algorithm, τ is the session key and $H(.)$ is a collision resistant hash function. Next, b is sent to the Crypto module while h will be used as a part of the final ciphertext.
2. On receiving b, the Crypto module runs an authenticated encryption algorithm CardAE, which could be any AE scheme using secret key K, which produces $C = AE_K(b)$ and sends it to the host. Note that the key K is only known to the Crypto module.
3. At the end, the host runs FinishAE by producing $C' = \langle C, h\rangle$ pair as the resulting authenticated encryption of M.

Similarly, the process of decryption is divided into 3 parts (shown in Fig. 1(b)):

1. For decryption, the host initially has the pair $C'= \langle C\|h\rangle$, out of which he sends C to the Crypto module.
2. Crypto module receives C and runs CardAD which is an authenticated decryption algorithm AD using secret key K, and outputs $b = AD_K(C)$ that will be sent to the host.
3. Finally, the host runs FinishAD (which we conveniently rename Open(h, b)). This deterministic algorithm Open(h, b) outputs M if (h, b) is a"valid" pair, otherwise it returns the invalid operator $\perp$. Open(h, b) first parses b into $\tau\|H(h)$, and then uses τ to decrypt h.

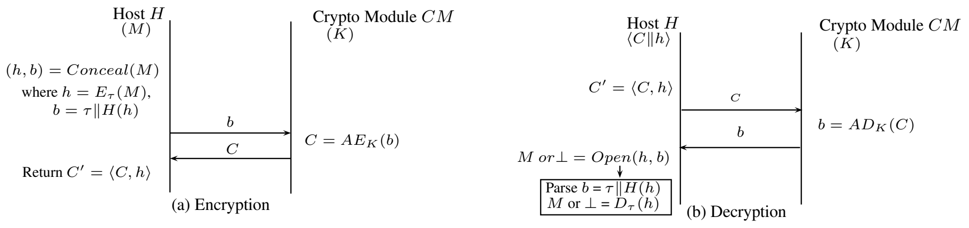

Fig. 1. Remote Key Authenticated Encyprion

2.2 Decrypt-Then-Mask Protocol

In [18], Fouque et al. proposed a generic construction called Decrypt-then-mask. Their construction doesn't affect encryption, only the decryption part is modified. They use a pseudorandom number generator to mask the plaintext blocks, to eliminate the need of storing plaintext blocks. Once the tag gets verified, the scheme returns the seed of the PRNG that has been used to mask the plaintext blocks. We use CM to denote cryptographic module and U to represent user of this device in further description of the scheme. For a given authenticated encryption scheme AE, the encryption and decryption component work as follows (shown in Fig. 2):

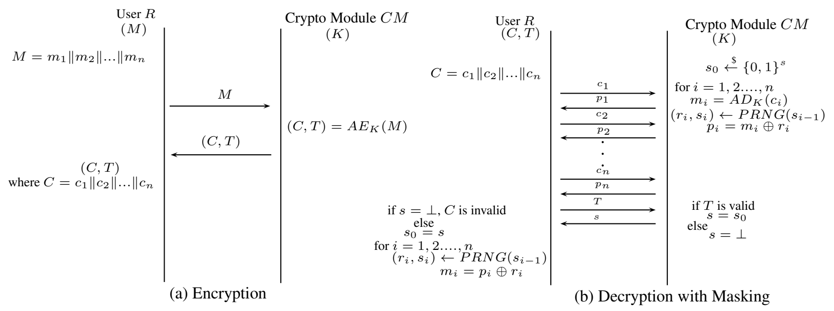

Fig. 2. Encryption and Decryption using Decrypt-Then-Mask Protocol: In (b)decryption, to represent it diagrammatically we assume $AD_K(c_i)$ returns the intermediate value m_i.

1. User U is given ciphertext-tag (C, T) pair,where $C = c_1\|c_2\|...\|c_n$, and he wants to decrypt it only if it has not been modified.
2. Crypto module CM executes the seed generation algorithm to generate a random seed s_0 for the PRNG and generates (r_1, s_1) = PRNG(s_0). User U sends first ciphertext block c_1 to CM. Crypto module CM initializes the tag computation and decrypts c_1 to generate $m_1 = AD_K(c_1)$ and masks m_1 to obtain $p_1 = m_1 \oplus r_1$ and sends it to U. In general, $AD_K(.)$ takes (C, T) pair as input and returns plaintext M or invalid operator $\perp$ (if the tag T is not valid). For better explaination, we assume $AD_K(c_i)$ returns the intermediate m_i.
3. U sends ciphertext block one by one to CM. Upon receiving each ciphertext block c_i, it updates the tag computation and decrypt c_i followed by masking $p_i = m_i \oplus r_i$ where r_i is generated as (r_i, s_i)= PRNG(s_{i-1}). Then p_i is returned back to the user U. This process repeats until all ciphertext blocks get processed.

4. Once the processing of all ciphertext blocks get done, U sends tag T to CM and the Crypto module CM checks validity of the tag. If the tag is valid then it returns s_0 to U, otherwise it returns invalid symbol $\perp$.
5. If U receives s_0, i.e., tag is valid and it decrypts $p_1\|p_2\|....\|p_n$. For $i = 1, 2...n$, it generates (r_i, s_i)= PRNG(s_{i-1}), and decrypt p_i as $m_i= p_i \oplus r_i$. Finally plaintext $M= m_1\|m_2\|....\|m_n$. If U receives $\perp$ then the tag is invalid and the plaintext M can not be generated.

3 Preliminaries

Definition 1 ***(Ideal Online Function)*** *Let **func**(A, B) be defined as set of all functions from set A to set B. Now, we define ideal online function $\$_{pad}$ as:*

$$\$_{pad}(N, A, M, flag) \rightarrow \begin{cases} (C, T) & \textit{if flag is 1,} \\ C & \textit{if flag is 0,} \end{cases}$$

where,

$pad(X)= X\|10^t$ where t is a non-negative integer s.t $|X\|10^t|= pr$ for $p > 0$ and some fixed r.
$N=$ nonce of r bit,
$A=$ Associated data, where $|A| < r$ and A'=pad(A),
$M=$ input message that can be partial or complete depending on the flag value,

$$flag = \begin{cases} 0\ ; & \textit{if user has queried partial message } M\ , \\ 1\ ; & \textit{if user has queried complete message } M\ . \end{cases}$$

In case $flag = 0$, we are assuming user has supplied incomplete message and wlog length of supplied message ($|M|$) is multiple of r.

$$M' = \begin{cases} m_0\| \ldots \|m_{n-1} = M, \textit{where } n = |M|/r \textit{ and } |m_i| = r & ; \textit{ if flag=0 }, \\ m_0\| \ldots \|m_{n-1} = pad(M) \textit{ s.t. } |m_i| = r \textit{ for } 0 \le i \le n-1 & ; \textit{ if flag=1 }. \end{cases}$$

$C = c_0\|c_1\| \ldots \|c_{n-1}$ where $c_j = g(N, A', m_0\| \ldots \|m_j)$ for $j = 0, \ldots, n-1$, and
$g \xleftarrow{\$} func(\{0,1\}^r \times \{0,1\}^r \times (\{0,1\}^r)^+, \{0,1\}^r)$,
$T = g'(N, A', M')$, where $g' \xleftarrow{\$} func(\{0,1\}^r \times \{0,1\}^r \times (\{0,1\}^r)^+, \{0,1\}^t)$.

Notice that above function is online: here prefixes of outputs remain the same if prefixes of the inputs remain constant. Furthermore, if we consider that nonce and associated data are unique for each function invocation in the ideal online function, then we acheive full privacy. This is because the inputs to g and g' will then be unique for each invocations, and since g and g' are functions randomly chosen from $func$, we get outputs that are independent and uniformly distributed.

Definition 2 ***(Privacy)*** *For a given authenticated encryption scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, we define privacy in terms of the ability of an adversary A to distinguish between the output of an encryption oracle $\mathcal{E}$ from the output of an ideal online function $\$_{pad}$. We define*

$$Adv^{priv}_{\mathcal{AE}}(A) = Pr[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}_K(.,.,.,.)} \Rightarrow 1] - Pr[A^{\$_{pad}(.,.,.,.)} \Rightarrow 1]$$

The oracle $\mathcal{E}_K$ on input $(N, A, M, flag)$, returns the C or (C, T) depending on the flag value whereas the oracle $\$_{pad}$ on input $(N, A, M, flag)$, works as defined in Def. 1. Here the advantage of an adversary A will be its ability to distinguish between the ciphertext outputs from the $\mathcal{E}_K$ and $\$_{pad}$.

Definition 3 ***(Authenticity)****. Here we give a notion of authenticity of the ciphertext tag pair of an authenticated encryption scheme. For a given authenticted encryption scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, let A be an adversary having an encryption oracle $\mathcal{E}_K$. We say that A forges if it outputs a (N, A, C, T) tuple where $\mathcal{D}_K(N, A, C, T) \neq INVALID$ and adversary A didn't ask a query $\mathcal{E}_K(N, A, M, flag)$ that resulted in response (C, T).*
Let $Exp^{Auth}_{\Pi}(A)$ be the forging experiment for a given $\mathcal{AE}$. Then the forging experiment is defined as follows:

1. *Adversary A can query encryption oracle $\mathcal{E}$, decryption oracle $\mathcal{D}$ atmost q_{enc}, q_{dec} times respectively.*
2. *All the query responses of encryption oracle $\mathcal{E}$, are stored in a set, say R. This R set contain (N, A, C, T) tuple.*

3. *If adversary A is able to generate a (N, A, C, T) tuple $\notin R$ that produces valid message M, then he wins and output is 1 otherwise output is 0, after trying q_{dec} number of queries.*

We say adversary wins i.e. he succeed in creating forgery if $Exp_{AE}^{Auth}(A) = 1$. So, the advantage of adversary in forging the scheme is defined as:

$$Adv_{AE}^{Auth}(A) = Pr[Exp_{AE}^{Auth}(A) = 1]$$

4 Our Construction: sp-AELM

A cryptographic module should not reveal any information about the plaintext until ciphertext-tag pair gets verified. Once the ciphertext gets verified, then crypto device is allowed to reveal the plaintext. However, a cryptographic module is likely to have some storage restriction. Therefore, it can not store a large plaintext M, verify the tag and output M only when M is valid. Traditional authenticated encryption schemes do not satisfy this usage scenario. We propose a new sponge based authenticated encryption scheme sp-AELM that considers this situation. sp-AELM uses a permutation as the underlying cryptographic primitive. Section 4.1 gives detail of the proposed scheme.

4.1 Description of sp-AELM

We now describe the new authenticated encryption scheme sp-AELM that addresses the low memory issue mentioned earlier. Schematic structure of sp-AELM is shown in Fig. 3. Our authenticated encryption scheme takes key K, nonce N, associated data A, plaintext M and $flag$ as inputs and returns C or (C, T) depending on the value of the $flag$, where C represents the ciphertext and T denotes tag or message authentication code (MAC). We denote cryptographic module by CM and the user of this module by U. The user U uses the crypto module CM for encryption, decryption and verification. The encryption and decryption procedures are given in Algorithm 1 and Algorithm 2 respectively.

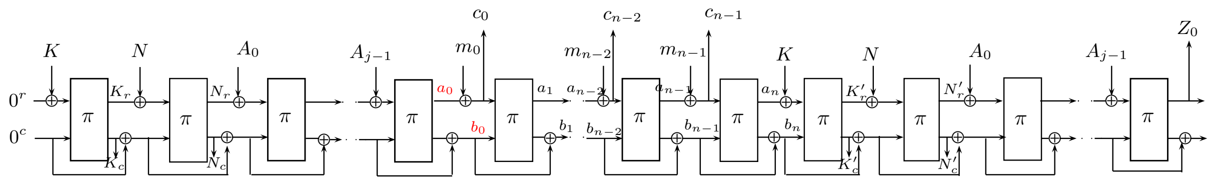

Fig. 3. sp-AELM Construction

4.1.1 Encryption

The algorithm for encryption, $\mathcal{E}_K(., ., ., .)$ is explained in Algorithm 1. It takes an input, the secret key K of r bits, nonce N of r bits and associated data A, plaintext M and a $flag$ value. This flag value can be 0 or 1, where 0 represents continue i.e. partial M is provided as an input and 1 represents stop i.e., M is complete, we can do the tag computation. Associatd data A is partitioned into r bit blocks$(A_0\| \ldots A_{j-1})$ and 10^* padding is used for last block if required. We use two initialization vectors IV_1 and IV_2 of size r and c respectively, each of which is initialized to zero. Further, flag value is checked, if it is 1, then the message M is padded using $Pad(M)$ and divided into n blocks $(M = m_0\|m_1\| \ldots m_{n-1})$, each block is of r bit (i.e. $|M|/r = n$). Otherwise it is just processed into r bit blocks. Algorithm iterates a fixed permutation $\pi : A \times B \to A \times B$, where $|A| = r$ *and* $|B| = c$. First $(IV_1 \oplus K)\|IV_2$ is given input to π and the output is $K_r\|K_c$. Then IV_2 is XORed with K_c(say this value X). The next input to π is $(K_r \oplus N)\|X$ and the corresponding output is $N_r\|N_c$. Then this N_c valus is XORed with X. Similarly associated data A is processed. After that message block $m_i's$ for $i = 0, 2, ..(n-1)$ get processed. Note that encryption is online here, we don't need to know message blocks in advance i.e., c_i can be calculated without prior knowledge of m_j for any $j > i$. Once all the message blocks are processed and $flag$ is 0, then $C = c_0\|c_1\|....\|c_{n-1}$ is return to the user and algorithm terminates. Otherwise K, N, A are processed again in a order followed by tag generation. Tag T is generated from Z_0 value. Z_0 is the initial r bits of final output of π and tag T is extraction of required initial bits of Z_0 (if tag is of 128 bit, then T is initial 128 bit of Z_0). Finally (C, T) pair is returned to the user, where $C = c_0\|c_1\|....\|c_{n-1}$.

Algorithm 1: Encryption $\mathcal{E}_K(N, A, M, flag)$

```
Initialization: $IV_1 = 0^r, IV_2 = 0^c$, $K_r\|K_c \xleftarrow{\$} \{0,1\}^b$,
$I_\pi = \{((IV_1 \oplus K)\|IV_2, K_r\|K_c)\}$
if $(flag = 1)$ then
    $M = m_0\|m_1\|.....\|m_{n-1}$, Where $|m_i| = r$ and $0 \le i < (n-1)$
    $Pad(M) = m_0\|m_1\|.....\|(m_{n-1}\|10^{r-(|m_{n-1}|+1)})$
else
    $M = m_0\|m_1\|...\|m_{n-1}$, Where $|m_i| = r$
    if $|m_{n-1}| < r$ then {return Invalid;}
$Pad(A) = A_0\|A_1 \ldots \|A_{j-1}$ s.t. $|A_i| = r$ and $0 \le i < (j-1)$
$x = K_r \oplus N$ , $w = K_c$
$N_r\|N_c = \pi(x\|w)$
$x = N_r$, $w = N_c \oplus w$
for $i = 0 \to j-1$ do
    $x'\|w' = \pi(x \oplus A_i\|w)$
    $w = w' \oplus w, x = x'$
$a_0 = x', b_0 = w'$
$c_0 = a_0 \oplus m_0$
for $i = 1 \to n-1$ do
    $x^{'}\|w^{'} = \pi(c_{i-1}\|b_{i-1})$
    $b_i = b_{i-1} \oplus w^{'}$
    $a_i = x^{'}$
    $c_i = a_i \oplus m_i$
$C = c_0\|c_1\|.....\|c_{n-1}$
if $(flag = 0)$ then
    Return $C$; Exit
$a_n\|b_n = \pi(c_{n-1}\|b_{n-1})$
$x = a_n, w = b_n$
$x = x \oplus K$
$K_r^{'}\|K_c^{'} = \pi(x\|w)$
$x = K_r^{'} \oplus N$ , $w = K_c^{'} \oplus w$
$N_r^{'}\|N_c^{'} = \pi(x\|w)$
$x = N_r^{'}$, $w = N_c^{'} \oplus w$
for $i = 0 \to j-1$ do
    $x'\|w' = \pi(x \oplus A_i\|w)$
    $w = w' \oplus w, x = x'$
$T = Z_0 = x$
Return $(C, T)$
```

Algorithm 2: Decryption $\mathcal{D}_K$ (N, A, C, T)

```
Initialization: $IV_1 = 0^r, IV_2 = 0^c$, $K_r\|K_c \xleftarrow{\$} \{0,1\}^b$,
$I_\pi = \{((IV_1 \oplus K)\|IV_2, K_r\|K_c)\}$
$|Pad(A)| = r$
$x = K_r \oplus N$ , $w = K_c$
$N_r\|N_c = \pi(x\|w)$
$x = N_r$, $w = N_c \oplus w$
for $i = 0 \to j-1$ do
    $x'\|w' = \pi(x \oplus A_i\|w)$
    $w = w' \oplus w, x = x'$
$a_0 = x, b_0 = w$
$m_0 = a_0 \oplus c_0$
for $i = 1 \to n-1$ do
    $x^{'}\|w^{'} = \pi(c_{i-1}\|b_{i-1})$
    $b_i = b_{i-1} \oplus w^{'}$
    $a_i = x^{'}$
    $m_i = a_i \oplus c_i$
$M = m_0\|m_1\|.....\|m_{n-1}$
$x = a_n, w = b_n$
$x = x \oplus K$
$K_r^{'}\|K_c^{'} = \pi(x\|w)$
$x = K_r^{'} \oplus N$ , $w = K_c^{'} \oplus w$
$N_r^{'}\|N_c^{'} = \pi(x\|w)$
$x = N_r^{'}$, $w = N_c^{'} \oplus w$
for $i = 0 \to j-1$ do
    $x'\|w' = \pi(x \oplus A_i\|w)$
    $w = w' \oplus w, x = x'$
$Z_0 = x$
if $(Z_0 == T)$ then
    Return $(a_0, b_0)$
else
    Return $\perp$
```

4.1.2 Decryption and Verification

The algorithm for decryption $\mathcal{D}_K(.,.,.,.)$ is given in Algorithm 2. The decryption process is similar to encryption. All the steps are same except for XORing with the message. The only difference is rather than XORing with m_i, we XOR with c_i. Once the message block m_i is evaluated we use that as input just like encryption. Here in this decryption algorithm, we just store (a_0, b_0) value, all $m_i's$ block not get stored anywhere. Finally tag Z_0 is calculated. Once the tag becomes available , the given tag T is compared with Z_0. If tag gets verified, i.e., $Z_0 = T$, then algorithm returns (a_0, b_0) value. Once the user get this value, he can easily recover plaintext M from C as m_0= $c_0 \oplus a_0$, $m_1 = c_1 \oplus \pi(c_0, b_0)$[initial r bit] and so on (explained in Algorithm 2). If tag doesn't get verified, then it returns invalid operator, i.e., plaintext cannot be calculated in this case. Fig. 4(b) shows the decryption protocol used in this new scheme.

5 Security Proof

In this section, we provide security proofs for the privacy and authenticity of spALEM. We have used recently evolved game playing technique proposed by Bellare and Rogaway in [5]. We prove security of sp-AELM in the ideal permutation model, where the underlying permutation is assumed to behave perfectly random. We also restrict our proof to the assumption that nonce N will always be generated different and randomly by the attacker. As it is not practically feasible by the algorithm to generate it randomly and differently everytime. To maintain the

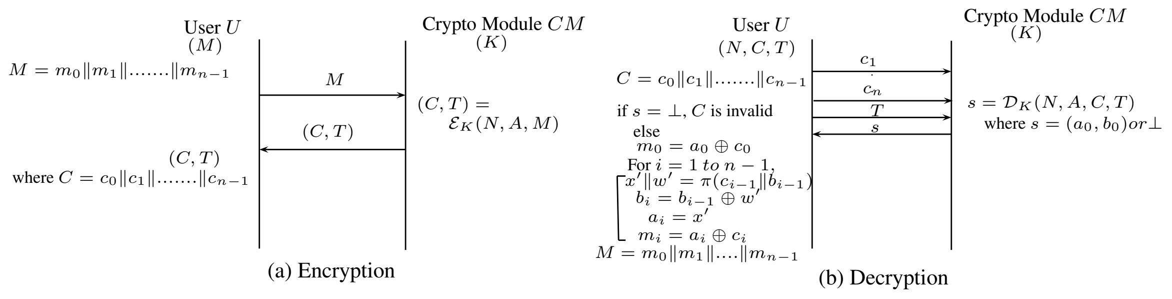
User U
(M)
$M = m_0\|m_1\|.......\|m_{n-1}$
Crypto Module CM
(K)
M
$(C,T) = \mathcal{E}_K(N,A,M)$
(C,T)
(C,T)
where $C = c_0\|c_1\|.......\|c_{n-1}$
(a) Encryption
User U
(N,C,T)
$C = c_0\|c_1\|.......\|c_{n-1}$
if $s = \perp$, C is invalid
else
$m_0 = a_0 \oplus c_0$
For $i = 1$ to $n-1$,
$x'\|w' = \pi(c_{i-1}\|b_{i-1})$
$b_i = b_{i-1} \oplus w'$
$a_i = x'$
$m_i = a_i \oplus c_i$
$M = m_0\|m_1\|....\|m_{n-1}$
Crypto Module CM
(K)
c_1
c_n
T
s
$s = \mathcal{D}_K(N,A,C,T)$
where $s = (a_0, b_0) or \perp$
(b) Decryption

Fig. 4. Encryption and Decryption protocol for sp-AELM

simplicity of proof, we have considered associated data of one block only. This can be further extended with minor changes in proof.

5.1 Privacy

We obtain an upper bound for the advatnage of the adversary who can distinguish the output of the proposed scheme with a random oracle in the ideal permutation model.

Theorem 1. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ denote the proposed Authenticated Encryption scheme with defined padding rule and an permutation π, which operates on b bits. The adversary A has given access to π, π^{-1}. Then the advantage of A relative to $\mathcal{E}$ is given by*

$$Adv_{\Pi}^{priv}(A) = Pr[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}_K(),\pi,\pi^{-1}} = 1] - Pr[K \xleftarrow{\$} \mathcal{K} : A^{\$_{pad}(),\pi,\pi^{-1}} = 1] \ .$$

$$Adv_{\Pi}^{priv}(A) \leq \frac{\sigma(\sigma-1)}{2^b+1} + \frac{\sigma(\sigma-1)}{2^c+1} + \frac{(q_\pi + q_{\pi^{-1}}).q_{enc}}{2^r} + \frac{2(q_\pi + q_{\pi^{-1}})\sigma}{2^c - \sigma} + \frac{q_\pi}{2^r} \ .$$

where $\$_{pad}$ is defined in Def. 1, σ is the maximum number of block calls to π, π^{-1} by encryption $\mathcal{E}$ and decryption $\mathcal{D}$ algorithm . q_{enc}, q_π and $q_{\pi^{-1}}$ are the maximum number of queries to encryption oracle Enc, π and π^{-1} oracle respectively by adversary A. $b(= r + c)$ is the size on which π permutation operates.

Proof. The advantage of the adversary is his ability to distinguish the proposed scheme from a random oracle. We used the game based framework to compute this advantage. We define a sequence of nine games from $G0$ to $G8$ given in supporting document along with this paper, where Game $G0$ represents the proposed scheme and $G8$ represents completely random output. Computations for probability difference between consecutive games are provided below.
Game G0: In this game encryption oracle perfectly simulates the proposed algorithm $AE - E^{\pi,pad}$ and π, π^{-1} oracle simulate an ideal permutation and its inverse.

$$Pr[A^{E_k,\pi,\pi^{-1}} = 1] = Pr[A^{G0} = 1]$$

Game G1: Game $G1$ is exactly same as game $G0$.

$$Pr[A^{G0} = 1] = Pr[A^{G1} = 1]$$

Game G2: Game G1 and G2 differs only when bad event occurs, otherwise, in absense of bad event, both games are same.

$$\mid Pr[A^{G1} = 1] - Pr[A^{G2} = 1] \mid \leq Pr[bad]$$

Bad event occurs when input to π and π^{-1} collide with the elements in set I_π. So, the probability of bad event will be equal to the probability of collision in π and π^{-1}. Let $Pr[coll]$ be the probability of collision in π and π^{-1}, then

$$Pr[bad] \leq Pr[coll]$$

Let $Pr[coll_i] = (i-1)/2^b$ be the probability that there is no collision till $i-1$ queries and it occurs in i^{th} query. Thus $Pr[coll]$ can be calculated as:

$$\begin{aligned} Pr[coll] &= Pr[coll_1 \vee coll_2 \vee ......coll_\sigma] \\ &\leq Pr[coll_1] + Pr[coll_2] + ... + Pr[coll_\sigma] \\ &\leq \frac{1}{2^b} + \frac{2}{2^b} + ... + \frac{\sigma-1}{2^b} \\ &\leq \frac{\sigma(\sigma-1)}{2^{b+1}}. \end{aligned}$$

Hence, the probability difference between $G1$ and $G2$ is

$$\mid Pr[A^{G1} = 1]\text{-}Pr[A^{G2} = 1] \mid \leq \frac{\sigma(\sigma-1)}{2^{b+1}}$$

Game G3: Game $G2$ and $G3$ are identical. In game $G3$, Enc oracle simulates behaviour of π, so it becomes independent of π. Hence, Game $G2$ and $G3$ are same from adversary point of view.

$$Pr[A^{G2} = 1] = Pr[A^{G3} = 1]$$

Game G4: Game $G3$ and $G4$ differ only when *bad* event occurs, otherwise both are exactly same. This *bad* event occurs if collision happens over c bits inside I_c set in $G4$.

$$\mid Pr[A^{G3} = 1] - Pr[A^{G4} = 1] \mid \leq Pr[bad]$$

And, $$Pr[bad] \leq Pr[collision\ over\ c\ bits]$$

$$\begin{aligned} Pr[collision\ over\ c\ bits] &\leq \frac{1}{2^c} + \frac{2}{2^c} + ... + \frac{\sigma-1}{2^c} \\ &\leq \frac{\sigma(\sigma-1)}{2^{c+1}} \end{aligned}$$

Hence, $$\mid Pr[A^{G3} = 1] - Pr[A^{G4} = 1] \mid \leq \frac{\sigma(\sigma-1)}{2^{c+1}}$$

Game G5: Game $G4$ and $G5$ are exactly same from adversarial point of view. We have created two new sets $I_\pi^{'}$ and $I_\pi^{''}$, where, $I_\pi^{'}$ keeps store for all the queries to encryption oracle and $I_\pi^{''}$ keeps store for all the queries to π and π^{-1} oracle. Now, set I_π is union of $I_\pi^{'}$ and $I_\pi^{''}$. This doesn't make any difference in functionality of game $G4$ and $G5$.

$$Pr[A^{G4} = 1] = Pr[A^{G5} = 1]$$

Game G6: Game $G5$ and $G6$ are same, except $G6$ makes an additional check in sets $I_\pi^{'}$ and $I_\pi^{''}$. But from adversary's perspective both games are exactly same.

$$Pr[A^{G5} = 1] = Pr[A^{G6} = 1]$$

Game G7:Game $G6$ and $G7$ will differ in presence of bad_1, bad_2 and bad event.

$$\mid Pr[A^{G6} = 1]\text{-}Pr[A^{G7} = 1] \mid \leq Pr[bad_1] + Pr[bad_2] + Pr[bad]$$

We are assuming here that q_π and $q_{\pi^{-1}}$ query already has been queried to π and π^{-1} oracle respectively. And maximum number of encryption queries are q_{enc}.
Calculation for bad_1 event : bad_1 is an event of having collision at the first block over r bits in set $I_\pi^{''}$ for $G6$, as c bits are fixed here. So,

$$Pr[bad_1] \leq Pr[collision\ over\ r\ bits]$$

And, $$Pr[collision\ over\ r\ bits] \leq \frac{(q_\pi + q_{\pi^{-1}}).q_{enc}}{2^r}$$

Calculation for bad_2 event : bad_2 is an event of having collision over c bits in set $I_\pi^{''}$ for $G6$, as r bits can be controlled by adversary (by changing message). And these c bits are generated randomly and different each time.

$$Pr[bad_2] \leq Pr[collision\ over\ c\ bits]$$

And, $$Pr[collision\ over\ c\ bits] \leq \frac{(q_\pi + q_{\pi^{-1}})\sigma}{2^c - \sigma}$$

Calculation for probability of *bad* event : Here, *bad* is an event of having collision in the set I''_π. This can happen either if collision occurs over c bit for a message M, as r bits can be controlled by an adversary or he is able to guess the key correctly. Hence,

$$Pr[bad] \leq Pr[correct\ key\ guess] + Pr[collision\ over\ c\ bits]$$

Here, $$Pr[collision\ over\ c\ bits] \leq \frac{(q_\pi + q_{\pi^{-1}})\sigma}{2^c - \sigma} \quad \text{(same as computed above)}$$

And, $$Pr[correct\ key\ guess] \leq \frac{q_\pi}{2^r}$$

So, $$\mid Pr[A^{G6} = 1]\text{-}Pr[A^{G7} = 1] \mid \leq \frac{(q_\pi + q_{\pi^{-1}}).q_{enc}}{2^r} + \frac{2(q_\pi + q_{\pi^{-1}})\sigma}{2^c - \sigma} + \frac{q_\pi}{2^r}$$

Game G8: In game $G7$ each ciphertext block is generated randomly, then concatenated and return to an adversary. Similarly Tag is also generated randomly. However, in Game $G8$, ciphertext and tag of desired length is selected as a random string and returned to an adversary. This move from $G7$ to $G8$ doesn't make any difference, as in both the games output is generated as completely random value. So,

$$Pr[A^{G7} = 1] = Pr[A^{G8} = 1]$$

Finally, using the fundamental lemma of game based framework,

$$\begin{aligned}
Adv_{\Pi}^{priv}(A) &= Pr[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}_K, \pi, \pi^{-1}} = 1] - Pr[K \xleftarrow{\$} \mathcal{K} : A^{\$_{pad}(), \pi, \pi^{-1}} = 1] \\
&= \mid \Pr[A^{G0} = 1] - Pr[A^{G8} = 1] \mid \\
&= \mid (Pr[A^{G0} = 1] - Pr[A^{G1} = 1]) \mid + \mid (Pr[A^{G1} = 1] - Pr[A^{G2} = 1]) \mid \\
&\quad + \mid (Pr[A^{G2} = 1] - Pr[A^{G3} = 1]) \mid + \mid (Pr[A^{G3} = 1] - Pr[A^{G4} = 1]) \mid \\
&\quad + \mid (Pr[A^{G4} = 1] - Pr[A^{G5} = 1]) \mid + \mid (Pr[A^{G5} = 1] - Pr[A^{G6} = 1]) \mid \\
&\quad + \mid (Pr[A^{G6} = 1] - Pr[A^{G7} = 1]) \mid + \mid (Pr[A^{G7} = 1] - Pr[A^{G8} = 1]) \mid \\
&\leq 0 + \frac{\sigma(\sigma - 1)}{2^b + 1} + 0 + \frac{\sigma(\sigma - 1)}{2^c + 1} + 0 + 0 \\
&\quad + \frac{(q_\pi + q_{\pi^{-1}}).q_{enc}}{2^r} + \frac{2(q_\pi + q_{\pi^{-1}})\sigma}{2^c - \sigma} + \frac{q_\pi}{2^r} .
\end{aligned}$$

5.2 Authenticity

In this section, we analyze the security of authenticity of the tag produced in our scheme. The forgery of an AE scheme is defined as the ability of an adversary A to generate a valid (N, A, C, T) tuple, without directly querying it to the encryption oracle. The adversary is allowed to make limited number of queries to encryption, decryption, π and π^{-1} oracles. For an AE scheme, we say the adversary A is successful in forging if it outputs a (N, A, C, T) tuple where $\mathcal{D}_K(N, A, C, T) \neq INVALID$ and adversary A didn't ask a query $\mathcal{E}_K(N, A, M, flag)$ that resulted in response (C, T) .

Theorem 2. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the proposed authenticated encryption scheme with defined padding rule (pad) and ideal permutation (π) which operate on $b(= r + c)$ bits. The adversary A is given access to Encryption oracle $\mathcal{E}$, Decryption oracle $\mathcal{D}$, π and π^{-1} oracle. Then $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is forgeable with the probability*

$$Pr[Exp_{\Pi}^{Auth}(A) = 1] \leq \frac{\sigma(\sigma - 1)}{2^{b+1}} + \frac{\sigma(\sigma - 1)}{2^c + 1} + \frac{q_{dec}}{2^{|T|}} + \frac{q_\pi}{2^r} .$$

where σ is maximum number of blocks to π, π^{-1} by encryption oracle $\mathcal{E}$, and decryption oracle $\mathcal{D}$, q_{dec} is the number of queries to decryption oracle, q_π is the number of queries to π oracle and $|T|$ is the tag length.

Proof: We used code base gaming framework to calculate the adversary advantage. Here we used the set of five games($G0', G1', G2', G3', G4'$) that is same as initial five games for privacy proof except that decryption oracle will also work here. Games are given in supporting document with this paper. Our goal is to bound the adversary's advantage which is defined as follows:

$$Adv_{\Pi}^{Auth}(A) = Pr[Exp_{\Pi}^{Auth}(A) = 1] \qquad (1)$$

Here, probability calculations will be same as that of initial five games of privacy explained in Section 5.1. So, we will directly use the results from there.
Calculations for $Pr[Exp_{G4'}^{Auth}(A=1)]$: In this game $G4'$, adversary gets a success in generating valid tag pair either by choosing tag randomly after trying q_{dec} number of queries or he guesses the key correctly. Hence adversary's ability to win in this game will be:

$$\begin{aligned} Pr[Exp_{G4}^{Auth}(A=1)] &\leq Pr[correct\ key\ guess] + Pr[random\ tag\ generation] \\ &\leq \quad \frac{q_\pi}{2^r} + \frac{q_{dec}}{2^{|T|}} \end{aligned}$$

On combining the results obtained in all the games, equation (1) becomes,

$$\begin{aligned} Adv_{\Pi}^{Auth}(A) &= Pr[Exp_{\Pi}^{Auth}(A)=1] \\ &\leq |\ Pr[Exp_{G1'}^{Auth}(A)=1] - Pr[Exp_{G2'}^{Auth}(A)=1]\ | + Pr[Exp_{G2'}^{Auth}(A)=1] \\ &\leq \frac{\sigma(\sigma-1)}{2^{b+1}} + Pr[Exp_{G3'}^{Auth}(A)=1] \\ &\leq \frac{\sigma(\sigma-1)}{2^{b+1}} + |\ Pr[Exp_{G3'}^{Auth}(A)=1] - Pr[Exp_{G4'}^{Auth}(A)=1]\ | + Pr[Exp_{G4'}^{Auth}(A)=1] \\ &\leq \frac{\sigma(\sigma-1)}{2^{b+1}} + \frac{\sigma(\sigma-1)}{2^{c+1}} + \frac{q_\pi}{(2^c-q_\pi)^2} + Pr[Exp_{G4'}^{Auth}(A)=1] \\ &\leq \frac{\sigma(\sigma-1)}{2^{b+1}} + \frac{\sigma(\sigma-1)}{2^{c+1}} + \frac{q_{dec}}{2^{|T|}} + \frac{q_\pi}{2^r}\ . \end{aligned}$$

$$Adv_{AE}^{Auth}(A) \leq 0 + \frac{\sigma(\sigma-1)}{2^b+1} + 0 + \frac{\sigma(\sigma-1)}{2^c+1} + \frac{q_\pi}{(2^c-q_\pi)^2} + \frac{1}{2^{|T|}}\ .$$

6 More variants of sp-AELM

This section presents two more variants of sp-AELM. The advantage of these variants over actual construction is that it doesn't require feed forward operation. Schematically construction of sp-AELM variants are shown in Fig. 5 and Fig. 6. These variants support the low memory constraint in a same way as in sp-AELM by storing only one intermediate state(shown using red line in Fig. 5) instead of storing complete text. Security proof for these constructions are not provided here due to space restrictions, although, it can be proved in same manner as sp-AELM. Intuitively, we can say these are secure as releasing intermediate state will not result for the adversary to gain any information about key. One can choose these variants over sp-AELM, when there is very limited amount of memory, which is not even capable of storing states required for feed-forward operation.

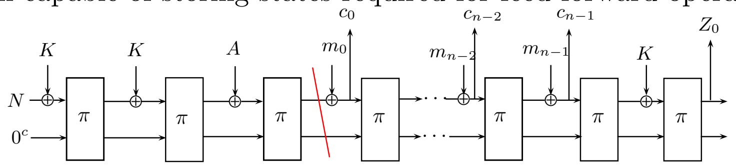

Fig. 5. sp-AELM variant 1

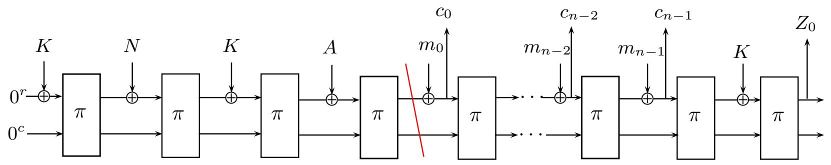

Fig. 6. sp-AELM variant 2

7 Analysis of Sponge based AE schemes submitted in CAESAR

In this section, we apply our newly proposed generalized technique, which stores only one intermediate state instead of entire plaintext during tag verification, on various sponge based schemes submitted to the CAESAR competition to determine their suitability for supporting devices having limited memory constraint. There are ten sponge based AE schemes submitted to CAESAR for the first round, out of which one scheme (CBEAM [26]) has been withdrawn. Currently there are nine sponge based AE schemes competing for the next round. In the following subsections we present the brief analysis of each of these 9 schemes after applying the same technique used in sp-AELM.

7.1 ARTEMIA [22]

Artemia is family of dedicated authenticated encryption scheme. It has two variants Artemia-256 which uses 512 bit permutation and Artemia-128 which uses 256 bit permutation in the JHAE mode(shown in Fig. 7).

While analyzing this mode, we find that it can not support low memory device constraint. Our analysis is based on the same technique proposed in sp-AELM i.e., storing only one intermediate state(shown using red line in Fig. 7) instead of storing whole message during decryption, can not be applied here. Using this intermediate state attacker can find the value of key as he already knows the value of T and can do the forward computation using C_i^s to get the state value after last π block, XORing this value with T will result in value of K.

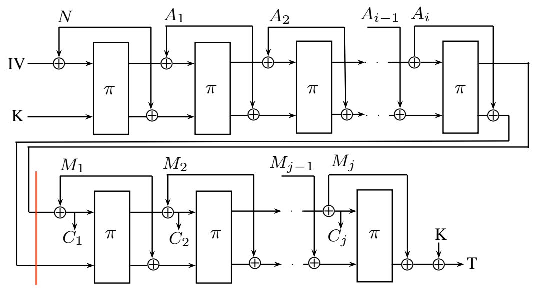

Fig. 7. JHAE Mode

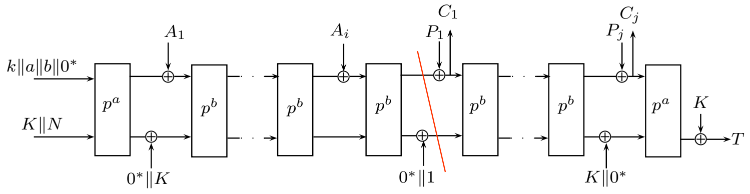

Fig. 8. ASCON

7.2 ASCON [12]

ASCON is a family of authenticated encryption designs $ASCON_{a,b} - k$. The family members are parameterized by the key length $k \leq 128$ and internal round numbers a and b. Each design specifies an authenticated encryption algorithm $\mathcal{E}_{k,a,b}$ and decryption algorithm $\mathcal{D}_{k,a,b}$.

On analyzing this scheme, we find out that it can support low memory devices by storing only one intermediate value(shown using red line in Fig. 8) instead of storing all decrypted blocks during decryption, without breaking the security of construction. This is due to the use of key at the end, which also prevents attcacker from doing forgery. Further, this intermediate value can be used to decrypt the message at user side.

7.3 ICEPOLE [29]

ICEPOLE is a family of authenticated ciphers with three parameters: key length, secret message number legth, nonce length. The construction of ICEPOLE is shown in Fig. 9.

Here, if we store intermediate state(shown using red line in Fig. 9) instead of whole decrypted text blocks to support low memory feature, it will result in revealing the key value, as attacker can do calculation in reverse direction to get the actual value of key.

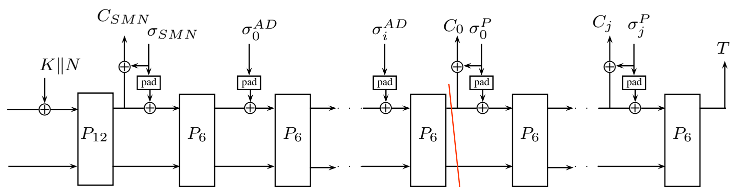

Fig. 9. ICEPOLE

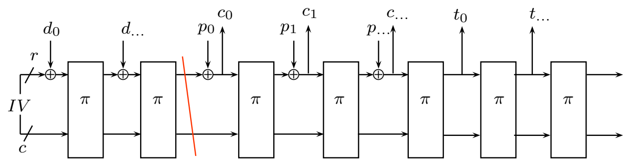

Fig. 10. STRIBOB. Key, nonce and padded associated data are represented by d_i and tag is givenn by t_i.

7.4 STRIBOB [27]

STRIBOB is an algorithm for authenticated encryption with associated data. It is built on basic sponge mode of authenticated encryption(shown in Fig. 10).It uses a 512 512 bit permutation π as its cryptographic foundation. Π in turn is built from 12 iterations of LPS transformation, interleaved with exclusive-or operation with round constants.

In STRIBOB, if we store only intermediate state(shown using red line in Fig. 10), instead of storing all $P_i^{'}s$, attacker can get the actual key value by using inverse permutation.

7.5 *Π*-Cipher [13]

Π-Cipher is parallel, incremental, nonce based, tag second-preimage resistant, authenticated encryption cipher with associated data. Its construction is shown in Fig. 11.

This AE scheme can not support limited memory constraint. Suppose if we store CIS'' and $ctr + i + 2$ values instead of storing all decrypted text blocks. Now using these intermediate values and known ciphertext values attacker can calculate the plaintext and Tag T''. However using this T'' he can not get the key due to the unavailability of Secret message number(SMN). Though it is secure against key recovery attack but forgery is possible here. As using T'', it is easy to generate a new ciphertext C and its corresponding valid tag T using a repeated nonce(SMN). This is due to the reason that T'' value will be same for a given key K and secret message number SMN.

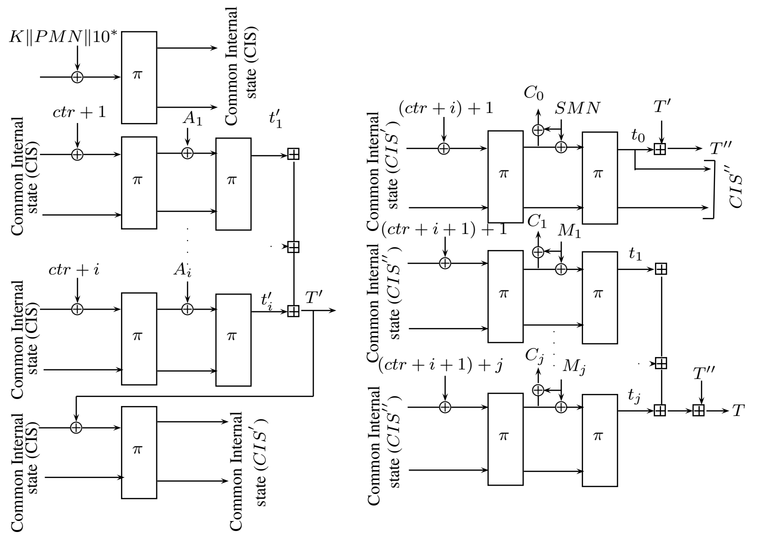

Fig. 11. Π Cipher

7.6 PRIMATE [17]

The authenticated encryption family PRIMATEs is defined by two parameters: the security level $s \in \{10, 15\}$ and mode of operation scheme $\in$ {GIBBON, HANUMAN, APE}. These modes are shown in Fig. 12, 13, 14.

Out of these three modes of operation of PRIMATE authenticated encryption family, only GIBBON can support low memory feature by storing only one intermediate state(shown using red line in Fig. 12), without revealing key to the attacker. HANUMAN can be easily broken to get the key, when storing only intermediate state as T is known to the attacker and can do the forward computation to get the state after last permutation block, XORing this value with T will result in finding key. Also, attacker will not be able to create forgery due to the use of key at the end. Similarly APE can also be broken, if we implement the low memory feature in it. As the intermediate state can be used to calculate the key by just applying permutation once and then XORing the last $|T|$ bits with known tag value T.

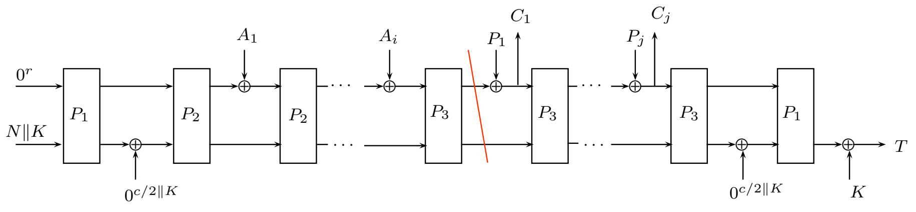

Fig. 12. GIBBON

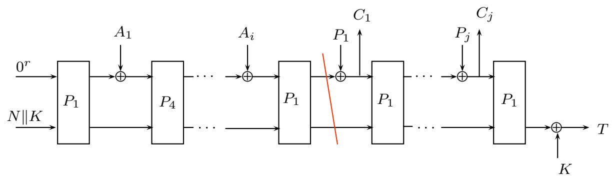

Fig. 13. HANUMAN

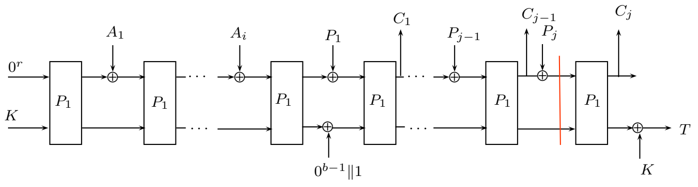

Fig. 14. APE

7.7 NORX [23]

NORX is an authenticated encryption scheme supporting an arbitrary parallelism degree, based on ARX primitives yet not using modular additions. NORX has a unique parallel architecture based on the monkeyDuplex construction [19]. Fig. 15 represents layout of NORX corresponding to $D = 1$. Here also, revealing the intermediate state(shown using red line in Fig. 15) will result in finding value of key by doing computation in reverse direction using inverse permutation, breaking the security of construction.

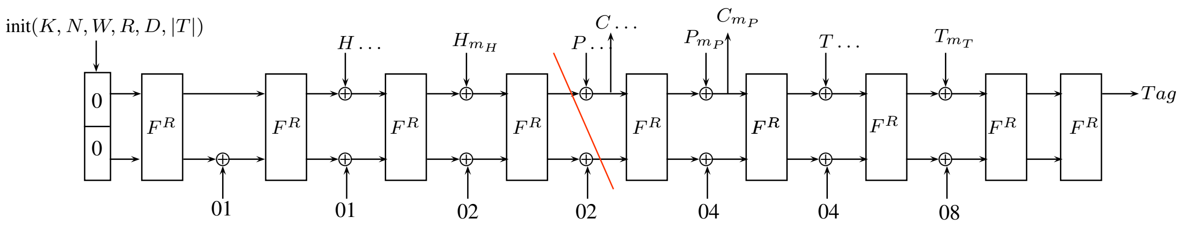

Fig. 15. NORX for D=1

7.8 Ketje [20]

Ketje is a set of two authenticated encryption functions with support for message associated data. Ketje builds on round-reduced versions of Keccak-f[400] and Keccak-f[200].The construction calling these permutations is MonkeyDuplex [19], a variant of the duplex construction [9]. The mode that runs on top of MonkeyDuplex is called MonkeyWrap. The MonkeyWrap differ from SpongeWrap as it is built on MonkeyDuplex construction rather than on Duplex, it updates a whole state of b bits using key and nonce together instead of updating only bitrate part (r bits) in SpongwWrap and uses a different MonkeyDuplex call when transitioning to tag generation phase.

Implementation of low memory constraint is not possible for this scheme. As shown in the Fig. 16, storing intermediate state(shown using red line in Fig. 16), will result in finding value of key K using inverse permutation.

7.9 Keyak [21]

Keyak is a set of four authenticated encryption functions with support for message associated data. It builds on round-reduced versions of the Keyak-f [800] and Keyak-f [1600] permutations. It uses the duplex construction [7] on top of one of these permutations. The mode that runs on duplex construction is the DuplexWrap which is almost similar to SpongeWrap. In addition, DuplexWrap defines an explicit forget call(calling it is optional) to ensure

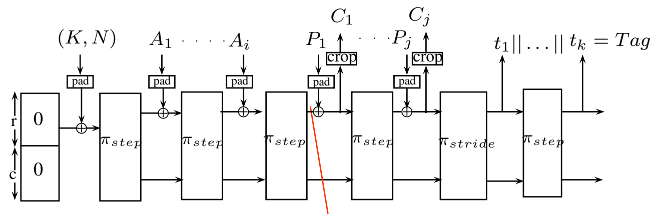

Fig. 16. MonkeyWrap mode of Authenticated Encryption

forward secrecy. Because of this forget call it is not possible to get the key from the intermediate state. But the forgery attack is possible by generating valid ciphertext tag pair for a different message using same key and nonce.

8 Conclusion

In this paper, we proposed a new generalized technique for AE schmes to support devices with limited memory. This new technique has been explained in this paper through a new sponge based AE scheme sp-AELM. sp-AELM can be used to support cryptographic modules having limited storage capabilities. We provided its security proof in an ideal permutation model using code based game playing framework for both privacy and authenticity. In addition to this, we also present two more variants of sp-AELM that serve the same purpose and are more efficient than sp-AELM. Further, we applied this newly introduced technique to all Sponge based AE schemes submitted to the CAESAR competition for determining their suitability to support devices with memory constraint. Our analysis shows that ASCON and one of the PRIMATEs instance namely GIBBON, can support limited memory constraint using this technique, while remaining schemes are not directly suitable for this scenario.

References

1. CAESAR: Competition for authenticated encryption: Security, applicability, and robustness, 2014. `http://competitions.cr.yp.to/caesar.html`.
2. Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. APE: Authenticated permutation-based encryption for lightweight cryptography. *IACR Cryptology ePrint Archive*, 2013:791, 2013.
3. Mihir Bellare and Chanathip Namprempre. Authenticated Encryption: Relations Among Notions and Analysis of the Generic Composition Paradigm. *J. Cryptol.*, 21(4):469–491, September 2008.
4. Mihir Bellare and Phillip Rogaway. Encode-Then-Encipher Encryption: How to Exploit Nonces or Redundancy in Plaintexts for Efficient Cryptography. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330. Springer, 2000.
5. Mihir Bellare and Phillip Rogaway. Code-Based Game-Playing Proofs and the Security of Triple Encryption. *IACR Cryptology ePrint Archive*, 2004:331, 2004.
6. Mihir Bellare, Phillip Rogaway, and David Wagner. EAX: A Conventional Authenticated-Encryption Mode. *IACR Cryptology ePrint Archive*, 2003:69, 2003.
7. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 320–337. Springer, 2011.
8. Guido Bertoni, Joan Daemen, Michal Peeters, and Gilles Van Assche. Cryptographic sponge functions, 2011. `http://sponge.noekeon.org/`.
9. Bertoni, Guido and Daemen, Joan and Peeters, Michaël and Van Assche, Gilles. Duplexing the Sponge: Single-pass Authenticated Encryption and Other Applications. In *Proceedings of the 18th International Conference on Selected Areas in Cryptography*, SAC'11, pages 320–337, 2012.
10. Matt Blaze. High-Bandwidth Encryption with Low-Bandwidth Smartcards. In Dieter Gollmann, editor, *FSE*, volume 1039 of *Lecture Notes in Computer Science*, pages 33–40. Springer, 1996.
11. Matt Blaze, Joan Feigenbaum, and Moni Naor. A Formal Treatment of Remotely Keyed Encryption. In Kaisa Nyberg, editor, *EUROCRYPT*, volume 1403 of *Lecture Notes in Computer Science*, pages 251–265. Springer, 1998.
12. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, Martin Schlaffer. Ascon v1. `http://competitions.cr.yp.to/round1/asconv1.pdf`.
13. Danilo Gligoroski, Hristina Mihajloska, Simona Samardjiska, Hakon Jacobsen, Mohamed El-Hadedy and Rune Erlend Jensen. PiCipher v1. `http://competitions.cr.yp.to/round1/picipherv1.pdf`.
14. Yevgeniy Dodis. Concealment and Its Applications to Authenticated Encryption. In Alexander W. Dent and Yuliang Zheng, editors, *Practical Signcryption*, Information Security and Cryptography, pages 149–173. Springer, 2010.
15. Morris J. Dworkin. Sp 800-38c. recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality. Technical report, Gaithersburg, MD, United States, 2004.

16. Elena Andreeva and Andrey Bogdanov and Atul Luykx and Bart Mennink and Nicky Mouha and Kan Yasuda. How to Securely Release Unverified Plaintext in Authenticated Encryption. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, pages 105–125, 2014.
17. Elena Andreeva, Begul Bilgin, Andrey Bogdanov, Atul Luykx, Florian Mendel , Bart Mennink, Nicky Mouha, Qingju Wang and Kan Yasuda . PRIMATEs v1. `http://competitions.cr.yp.to/round1/primatesv1.pdf`.
18. Pierre-Alain Fouque, Antoine Joux, Gwenaëlle Martinet, and Frédéric Valette. Authenticated On-Line Encryption. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 145–159. Springer, 2003.
19. Guido Bertoni and Joan Daemen and Michal Peeters and Gilles Van Assche. G.V.: Permutationbased encryption, authentication and authenticated encryption, 2012.
20. Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche, Ronny Van Keer. Ketje v1. `http://competitions.cr.yp.to/round1/ketjev11.pdf`.
21. Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche, Ronny Van Keer. Ketje v1. `http://keyak.noekeon.org/Keyak-1.2.pdf`.
22. Javad Alizadeh, Mohammad Reza Aref and Nasour Bagheri. Artemia v1. `http://competitions.cr.yp.to/round1/artemiav1.pdf`.
23. Samuel Neves Jean-Philippe Aumasson, Philipp Jovanovic. NORX: Parallel and Scalable AEAD, 2014. `https://norx.io/`.
24. Tadayoshi Kohno, John Viega, and Doug Whiting. CWC: A High-Performance Conventional Authenticated Encryption Mode. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 408–426. Springer, 2004.
25. Stefan Lucks. On the Security of Remotely Keyed Encryption. In Eli Biham, editor, *FSE*, volume 1267 of *Lecture Notes in Computer Science*, pages 219–229. Springer, 1997.
26. Markku-Juhani O. Saarinen. The CBEAMr1 Authenticated Encryption Algorithm. `http://competitions.cr.yp.to/round1/cbeamr1.pdf`.
27. Markku-Juhani O. Saarinen. The STRIBOBr1 Authenticated Encryption Algorithm. `http://competitions.cr.yp.to/round1/stribobr1.pdf`.
28. David A. McGrew and John Viega. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004.
29. Pawel Morawiecki, Kris Gaj, Ekawat Homsirikamol, Krystian Matusiewicz, Josef Pieprzyk, Marcin Rogawski, Marian Srebrny, and Marcin Wojcik. ICEPOLE v1. `http://competitions.cr.yp.to/round1/icepolev1.pdf`.
30. Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, 2003.
31. Hongjun Wu and Bart Preneel. AEGIS: A Fast Authenticated Encryption Algorithm. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography*, volume 8282 of *Lecture Notes in Computer Science*, pages 185–201. Springer, 2013.

A sp-AELM variant supporting Intermediate tag

This section shows the sp-AELM variant that support intermediate tag generation and fast rejection of forgery for the long messages. Pictorial view is shown on Fig. 17. In this variant message $M(=m_1\|m_2\ldots\|m_{n-1})$ is divided into small packets$(m_1..m_i, m_{i+1}..m_j,\ldots,m_k..m_{n-1})$and individual tag is caluclated for every packet. Then ciphertext and tag for each packet is sent to receiver. At the receiver side ciphertext for first packet gets decrypted and tag computation id done. If computed tag for first packet matches with received tag then only further computation proceeds otherwise message gets rejected at the early stage only. During decryption only one intermediate state(shown by red line in Fig. 17) is stored instead of storing all decrypted text blocks and this intermediate state is returned to the user once complete message gets verified. Now, user can find the actual plaintext using this received intermediate state.

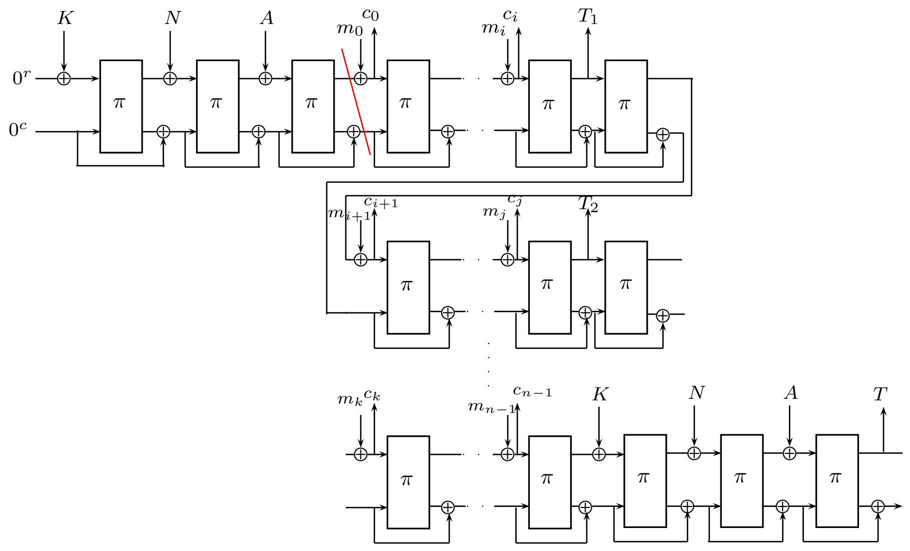

Fig. 17. sp-AELM variant with Intermediate tag generation

B Games for Privacy Proof

Game $\boxed{\text{G1}}$ and Game G2 : Initialise $IV_1 = 0^r, IV_2 = 0^c$, $K \xleftarrow{\$} \{0,1\}^r$, $K_r\|K_c \xleftarrow{\$} \{0,1\}^b$, $I_\pi = \{((IV_1 \oplus K)\|IV_2, K_r\|K_c)\}$

On Encryption-Query $(N, A, M, flag)$
Same as Game 0

On π-Query m,where $m \in \{0,1\}^b$

```
1. let $(x||w)$=$m$,where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. if $(m,v)\in I_\pi$ then return $v$
3. $v \xleftarrow{\$} \{0,1\}^b$
4. if $\exists$ $m'$ s.t $(m',v)\in I_\pi$, then bad←true and
   $\boxed{v \xleftarrow{\$} \{0,1\}^b \setminus \{v : (*,v) \in I_\pi\}}$, where $* \in \{0,1\}^b$
5. $I_\pi = I_\pi \bigcup \{(m,v)\}$
6. return $v$;
```

On π^{-1}-Query v, where $v \in \{0,1\}^b$

```
1. let $(v_1||v_2)$=$m$,where $v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$,
2. if $(m,v)\in I_\pi$ then return $m$
3. $m \xleftarrow{\$} \{0,1\}^b$
4. if $\exists$ $v'$ s.t $(m,v')\in I_\pi$, then bad←true and
   $\boxed{m \xleftarrow{\$} \{0,1\}^b \setminus \{m : (m,*) \in I_\pi\}}$, where $* \in \{0,1\}^b$
5. $I_\pi = I_\pi \bigcup \{(m,v)\}$
6. return $m$;
```

Fig. 19. Game G1 and Game G2

Game G0: Initialise $IV_1 = 0^r, IV_2 = 0^c$, $K \xleftarrow{\$} \{0,1\}^r$, $K_r \| K_c \xleftarrow{\$} \{0,1\}^b$, $I_\pi = \{((IV_1 \oplus K)\|IV_2, K_r\|K_c)\}$

On Encryption-Query $(N, A, M, flag)$

```
 1. if ($flag = 1$) then
        $M = m_0\|m_1\|.....\|m_{n-1}$, Where $|m_i| = r$ and
        $0 \leq i < (n-1)$
        $Pad(M) = m_0\|m_1\|.....\|(m_{n-1}\|10^{r-(|m_{n-1}|+1)})$
    else
        $M = m_0\|m_1\|...\|m_{n-1}$, Where $|m_i| = r$
        if $|m_{n-1}| < r$ then {return Invalid;}
 2. $|Pad(N)| = |Pad(A)| = r$
 3. $x = K_r \oplus N$ , $w = K_c$
 4. $N_r\|N_c = \pi(x\|w)$
 5. $x = N_r \oplus A$, $w = N_c \oplus w$
 6. $A_r\|A_c = \pi(x\|w)$
 7. $x = A_r, w = A_c \oplus w$
 8. $a_0 = x, b_0 = w$
 9. $c_0 = a_0 \oplus m_0$
10. for $i = 1 \to n$ do
        $x^{'}\|w^{'} = \pi(c_{i-1}\|b_{i-1})$
        $b_i = b_{i-1} \oplus w^{'}$, $a_i = x^{'}$
        $c_i = a_i \oplus m_i$
11. $C = c_0\|c_1\|.....\|c_n$
12. if ($flag = 0$) then
        return $C$.
13. $x = a_n, w = b_n$
14. $x = x \oplus K$
15. $K_r^{'}\|K_c^{'} = \pi(x\|w)$
16. $x = K_r^{'} \oplus N$ , $w = K_c^{'} \oplus w$
17. $N_r\|N_c = \pi(x\|w)$
18. $x = N_r \oplus A$, $w = N_c \oplus w$
19. $A_r\|A_c = \pi(x\|w)$
20. $x = A_r, w = A_c \oplus w$
21. $T = z_0 = x$
22. return $(C, T)$
```

On π-Query m, where $m \in \{0,1\}^b$

```
1. let $(x||w) = m$, where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. if $(m, v) \in I_\pi$ then return $v$
3. $v \xleftarrow{\$} \{0,1\}^b$
4. if $\exists\ m'$ s.t $(m', v) \in I_\pi$, then $v \xleftarrow{\$} \{0,1\}^b \setminus \{v : (*, v) \in I_\pi\}$,
   where $* \in \{0,1\}^b$
5. $I_\pi = I_\pi \bigcup \{(m, v)\}$
6. return $v$;
```

On π^{-1}-Query $v = \{v_1\|v_2\}$. where
$v_1 \in \{0,1\}^r,\ v_2 \in \{0,1\}^c,\ v \in \{0,1\}^b$

```
1. if $(m, v) \in I_\pi$ then return $m$
2. $m \xleftarrow{\$} \{0,1\}^b$
3. if $\exists\ v'$ s.t $(m, v') \in I_\pi$, then
   $m \xleftarrow{\$} \{0,1\}^b \setminus \{m : (m, *) \in I_\pi\}$, where $* \in \{0,1\}^b$
4. $I_\pi = I_\pi \bigcup \{(m, v)\}$
5. return $m$;
```

Fig. 18. Game G0

Game G3 and Game $\boxed{\textbf{G4}}$: Initialise $IV_1 = 0^r, IV_2 = 0^c$, $K \xleftarrow{\$} \{0,1\}^r$, $K_r\|K_c \xleftarrow{\$} \{0,1\}^b$, $I_\pi = \{((IV_1 \oplus K)\|IV_2, K_r\|K_c)\}$, $I_c = \{w\}$

```
On Encryption-Query $(N, A, M, flag)$

1. if $(flag = 1)$ then
       $M = m_0\|m_1\|.....\|m_{n-1}$, Where $|m_i| = r$ and
       $0 \le i < (n-1)$
       $Pad(M) = m_0\|m_1\|.....\|(m_{n-1}\|10^{r-(|m_{n-1}|+1)})$
   else
       $M = m_0\|m_1\|...\|m_{n-1}$, Where $|m_i| = r$
       if $|m_{n-1}| < r$ then {return Invalid;}
2. $|Pad(N)| = |Pad(A)| = r$
3. $x = IV_1, w = IV_2$
4. $x = K_r \oplus N$ ,$w = K_c$
5. if $\exists v\ s.t.(x\|w, v) \in I_\pi$, then
       $N_r\|N_c = v$
   else
       $N_r\|N_c \xleftarrow{\$} \{0,1\}^b$
       $\boxed{\text{if } (N_c \oplus w) \in I_c \text{ then } bad \leftarrow true}$
       $\boxed{N_c \xleftarrow{\$} \{0,1\}^c \setminus \{N_c^{'} : (N_c^{'} \oplus w) \in I_c\}}$
       $I_\pi = I_\pi \cup \{(x\|w, N_r\|N_c)\}$
6. $x = N_r \oplus A$ ,$w = N_c \oplus w$
7. $I_c = I_c \cup \{w\}$
8. if $\exists v\ s.t.\ (x\|w, v) \in I_\pi$, then
       $A_r\|A_c = v$
   else
       $A_r\|A_c \xleftarrow{\$} \{0,1\}^b$
       $\boxed{\text{if } (A_c \oplus w) \in I_c \text{ then } bad \leftarrow true}$
       $\boxed{A_c \xleftarrow{\$} \{0,1\}^c \setminus \{A_c^{'} : (A_c^{'} \oplus w) \in I_c\}}$
       $I_\pi = I_\pi \cup \{(x\|w, A_r\|A_c)\}$
9. $x = A_r$ ,$w = A_c \oplus w$
10. $I_c = I_c \cup \{w\}$
11. $a_0 = x, b_0 = w$
12. $c_0 = a_0 \oplus m_0$
13. for $i = 1 \to n$ do
        if $\exists v\ s.t.\ (c_{i-1}\|b_{i-1}, v) \in I_\pi$, then
            $x^{'}\|w^{'} = v$
        else
            $x^{'}\|w^{'} \xleftarrow{\$} \{0,1\}^b$
            $\boxed{\text{if } (b_{i-1} \oplus w^{'}) \in I_c \text{ then } bad \leftarrow true}$
            $\boxed{w^{'} \xleftarrow{\$} \{0,1\}^c \setminus \{w^{''} : (b_{i-1} \oplus w^{''}) \in I_c\}}$
            $I_\pi = I_\pi \cup \{(c_{i-1}\|b_{i-1}, x^{'}\|w^{'})\}$
        $b_i = b_{i-1} \oplus w^{'}$
        $I_c = I_c \cup \{b_i\}$
        $a_i = x^{'}$
        $c_i = a_i \oplus m_i$
14. $C = c_0\|c_1\|.....\|c_n$
15. if $(flag = 0)$ then
        return C.
16. $x = a_n, w = b_n$
17. $x = x \oplus K$
```

```
Continue...

18. if $\exists v\ s.t.(x\|w, v) \in I_\pi$, then
        $K_r^{'}\|K_c^{'} = v$
    else
        $K_r^{'}\|K_c^{'} \xleftarrow{\$} \{0,1\}^b$
        $\boxed{\text{if } (w \oplus K_c^{'}) \in I_c \text{ then } bad \leftarrow true}$
        $\boxed{K_c^{'} \xleftarrow{\$} \{0,1\}^c \setminus \{K_c^{''} : (w \oplus K_c^{''}) \in I_c\}}$
        $I_\pi = I_\pi \cup \{(x\|w, K_r^{'}\|K_c^{'})\}$
19. $x = K_r^{'} \oplus N$ , $w = K_c^{'} \oplus w$
20. $I_c = I_c \cup \{w\}$
21. Repeat step from 5 to 8.
22. $z_0 = x$
23. $T = z_0$
24. return $(C, T)$
```

```
On $\pi$-Query $m$, where $m \in \{0,1\}^b$

1. let $(x||w) = m$, where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. if $\exists$ $\{v_1, v_2, ...v_t\}$ s.t. $(m, v_i) \in I_\pi$ then return
   $v \xleftarrow{\$} \{v_1, v_2, ...v_t\}$
3. else $v \xleftarrow{\$} \{0,1\}^b$
4. $I_\pi = I_\pi \bigcup \{(m, v)\}$
5. return $v$;
```

```
On $\pi^{-1}$-Query $v = \{v_1||v_2\}$. where
$v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$, $v \in \{0,1\}^b$

1. if $\exists$ $\{m_1, m_2, ...m_t\}$ s.t. $(m_i, v) \in I_\pi$ then return
   $m \xleftarrow{\$} \{m_1, m_2, ...vm_t\}$
2. else $m \xleftarrow{\$} \{0,1\}^b$
3. $I_\pi = I_\pi \bigcup \{(m, v)\}$
4. return $m$;
```

Fig. 20. Game G3 and Game G4

Game G5 and Game $\boxed{\textbf{G6}}$: Initialise $IV_1 = 0^r, IV_2 = 0^c$, $K \xleftarrow{\$} \{0,1\}^r$, $K_r\|K_c \xleftarrow{\$} \{0,1\}^b$, $I'_\pi = \{((IV_1 \oplus K)\|IV_2, K_r\|K_c)\}$, $I''_\pi = \phi$, $I_\pi = I'_\pi \cup I''_\pi$, $I_c = \{K_c\}$, $I'_c = \phi$

On Encryption-Query $(N, A, M, flag)$

```
1. if $(flag = 1)$ then
     $M = m_0\|m_1\|.....\|m_{n-1}$, Where $|m_i| = r$ and
     $0 \le i < (n-1)$
     $Pad(M) = m_0\|m_1\|.....\|(m_{n-1}\|10^{r-(|m_{n-1}|+1)})$
   else
     $M = m_0\|m_1\|...\|m_{n-1}$, Where $|m_i| = r$
     if $|m_{n-1}| < r$ then {return Invalid;}
2. $|Pad(N)| = |Pad(A)| = r$
3. $x = IV_1, w = IV_2$
4. $x = K_r \oplus N$ ,$w = K_c$
5. if $\exists v$ s.t.$(x\|w, v) \in I_\pi$, then
     $\boxed{\text{if } \exists v \text{ s.t. } (x\|w, v) \in I''_\pi, \text{ then } bad_1 \leftarrow true}$
     $N_r\|N_c = v$
   else
     $N_r\|N_c \xleftarrow{\$} \{0,1\}^b$
     if $((N_c \oplus w) \in I_c$ OR $N_c \in I'_c)$, then
       $N_c \xleftarrow{\$} \{0,1\}^c \setminus \{N'_c : (N'_c \oplus w) \in I_c, N'_c \in I'_c\}$,
     $I'_\pi = I'_\pi \cup \{(x\|w, N_r\|N_c)\}$, $I_\pi = I_\pi \cup I'_\pi$
6. $x = N_r \oplus A$ ,$w = N_c \oplus w$
7. $I_c = I_c \cup \{w\}$, $I'_c = I'_c \cup \{N_c\}$
8. if $\exists v$ s.t.$(x\|w, v) \in I_\pi$, then
     $\boxed{\text{if } \exists v \text{ s.t. } (x\|w, v) \in I''_\pi, \text{ then } bad_2 \leftarrow true}$
     $A_r\|A_c = v$
   else
     $A_r\|A_c \xleftarrow{\$} \{0,1\}^b$
     if $((A_c \oplus w) \in I_c$ OR $A_c \in I'_c)$ then
       $A_c \xleftarrow{\$} \{0,1\}^c \setminus \{A'_c : (w \oplus A'_c) \in I_c, A'_c \in I'_c\}$
       $I'_\pi = I'_\pi \cup \{(x\|w, A_r\|A_c)\}$, $I_\pi = I_\pi \cup I'_\pi$
9. $x = A_r$ ,$w = A_c \oplus w$
10. $I_c = I_c \cup \{w\}$, $I'_c = I'_c \cup \{A_c\}$
11. $a_0 = x, b_0 = w$
12. $c_0 = a_0 \oplus m_0$
13. for $i = 1 \to n$ do
      if $\exists v$ s.t. $(c_{i-1}\|b_{i-1}, v) \in I_\pi$, then
        $\boxed{\text{if } \exists v \text{ s.t. } (c_{i-1}\|b_{i-1}, v) \in I''_\pi, \text{ then } bad_2 \leftarrow true}$
        $x'\|w' = v$
      else
        $x'\|w' \xleftarrow{\$} \{0,1\}^b$
        if $((b_{i-1} \oplus w') \in I_c$ OR $w' \in I'_c)$ then
          $w' \xleftarrow{\$} \{0,1\}^c \setminus \{w'' : (b_{i-1} \oplus w'') \in I_c, w'' \in I'_c\}$
        $I'_\pi = I'_\pi \cup \{(c_{i-1}\|b_{i-1}, x'\|w')\}$, $I_\pi = I_\pi \cup I'_\pi$
      $b_i = b_{i-1} \oplus w'$
      $I_c = I_c \cup \{b_i\}$, $I'_c = I'_c \cup \{w'\}$
      $a_i = x'$ , $c_i = a_i \oplus m_i$
14. $C = c_0\|c_1\|.....\|c_n$
15. if $(flag = 0)$ then
      return C.
16. $x = a_n, w = b_n$
17. $x = x \oplus K$
```

Continue..

```
18. if $\exists v$ s.t. $(x\|w, v) \in I_\pi$, then
      $\boxed{\text{if } \exists v \text{ s.t. } (x\|w, v) \in I''_\pi, \text{ then } bad_2 \leftarrow true}$
      $K'_r\|K'_c = v$
    else
      $K'_r\|K'_c \xleftarrow{\$} \{0,1\}^b$
      if $((w \oplus K'_c) \in I_c$ OR $w' \in I'_c)$, then
        $K'_c \xleftarrow{\$} \{0,1\}^c \setminus \{K''_c : (w \oplus K''_c) \in I_c,\ K''_c \in I'_c\}$
      $I'_\pi = I'_\pi \cup \{(x\|w, K'_r\|K'_c)\}$, $I_\pi = I_\pi \cup I'_\pi$
19. $x = K'_r \oplus N$ , $w = K'_c \oplus w$
20. $I_c = I_c \cup \{w\}$, $I'_c = I'_c \cup \{K'_c\}$
21. Repeat step from 5 to 8.
22. $z_0 = x$
23. $T = z_0$
24. return $(C, T)$
```

On π-Query m, where $m \in \{0,1\}^b$

```
1. let $(x||w) = m$, where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. if $\exists v$ s.t. $(x\|w, v) \in I'_\pi$, then $\boxed{bad \leftarrow true}$
     return $v$;
3. if $\exists$ $\{v_1, v_2, ...v_t\}$ s.t. $(m, v_i) \in I''_\pi$ then
     return $v \xleftarrow{\$} \{v_1, v_2, ...v_t\}$
4. else $v \xleftarrow{\$} \{0,1\}^b$
5. $I''_\pi = I''_\pi \bigcup \{(m, v)\}$
6. $I_\pi = I_\pi \cup I''_\pi$
7. return $v$;
```

On π^{-1}-Query $v = \{v_1||v_2\}$. where
$v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$, $v \in \{0,1\}^b$

```
1. if $\exists m$ s.t. $(m, v) \in I'_\pi$, then $\boxed{bad \leftarrow true}$
     return $m$
2. if $\exists$ $\{m_1, m_2, ...m_t\}$ s.t. $(m_i, v) \in I''_\pi$ then
     return $m \xleftarrow{\$} \{m_1, m_2, ..., m_t\}$
3. else $m \xleftarrow{\$} \{0,1\}^b$
4. $I''_\pi = I''_\pi \bigcup \{(m, v)\}$
5. $I_\pi = I_\pi \cup I''_\pi$
6. return $m$;
```

Fig. 21. Game G5 and Game G6

Game G7: Initialise $I_\pi = \emptyset$, $IV_1 = 0^r$ $IV_2 = 0^c$, $K \xleftarrow{\$} \{0,1\}^r$, $I_m = \phi$

On Encryption-Query $(N, A, M, flag)$

```
1. if ($flag = 1$) then
       $M = m_0\|m_1\|.....\|m_{n-1}$, Where $|m_i| = r$ and
       $0 \le i < (n-1)$
       $Pad(M) = m_0\|m_1\|.....\|(m_{n-1}\|10^{r-(|m_{n-1}|+1)})$
   else
       $M = m_0\|m_1\|...\|m_{n-1}$, Where $|m_i| = r$
       if $|m_{n-1}| < r$ then {return Invalid;}
2. $|Pad(N)| = |Pad(A)| = r$
3. $x = IV_1, w = IV_2$
4. if $(flag = 0)$ then
       if $\exists C'$ $s.t.(N, A, m_0\| \dots \|m_j, C') \in I_m$ where
       $0 \le j < (n-1)$ then
           $c_0\| \dots \|c_j \leftarrow C'$
           for $i = (j+1) \to (n-1)$ do
               $c_i \xleftarrow{\$} \{0,1\}^r$
           $C = c_0\| \dots \|c_j\|c_{j+1} \dots \|c_{n-1}$
       else
           for $i = 0 \to n-1$ do
               $c_i \xleftarrow{\$} \{0,1\}^r$
           $C = c_0\| \dots \|c_{n-1}$
       $I_m = I_m \cup \{(N, A, M, C)\}$
       return $C$
5. for $i = 0 \to n-1$ do
       $c_i \xleftarrow{\$} \{0,1\}^r$
6. $C = c_0\| \dots \|c_{n-1}$
7. $T \xleftarrow{\$} \{0,1\}^r$
8. return $(C, T)$
```

On π-Query m, where $m \in \{0,1\}^b$

```
1. let $(x||w) = m$, where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. if $(m, v) \in I_\pi$ then return $v$
3. $v \xleftarrow{\$} \{0,1\}^b$
4. if $\exists$ $m'$ s.t $(m', v) \in I_\pi$, then $v \xleftarrow{\$} \{0,1\}^b \setminus \{v : (*, v) \in I_\pi\}$,
   where $* \in \{0,1\}^b$
5. $I_\pi = I_\pi \bigcup \{(m, v)\}$
6. return $v$;
```

On π^{-1}-Query $v = \{v_1||v_2\}$. where
$v_1 \in \{0,1\}^r,\ v_2 \in \{0,1\}^c,\ v \in \{0,1\}^b$

```
1. if $(m, v) \in I_\pi$ then return $m$
2. $m \xleftarrow{\$} \{0,1\}^b$
3. if $\exists$ $v'$ s.t $(m, v') \in I_\pi$, then
   $m \xleftarrow{\$} \{0,1\}^b \setminus \{m : (m, *) \in I_\pi\}$, where $* \in \{0,1\}^b$
4. $I_\pi = I_\pi \bigcup \{(m, v)\}$
5. return $m$;
```

Fig. 22. Game G7

C Games for Authenticity Proof

Game $G0'$: Initialize $IV_1 = 0^r, IV_2 = 0^c$, $K \xleftarrow{\$} \{0,1\}^r$, $K_r \| K_c \xleftarrow{\$} \{0,1\}^b$

On Encryption-Query $(N, A, M, flag)$

```
1. if $(flag = 1)$ then
       $M = m_0 \| m_1 \| ..... \| m_{n-1}$, Where $|m_i| = r$ and
       $0 \leq i < (n-1)$
       $Pad(M) = m_0 \| m_1 \| ..... \| (m_{n-1} \| 10^{r-(|m_{n-1}|+1)})$
   else
       $M = m_0 \| m_1 \| ... \| m_{n-1}$, Where $|m_i| = r$
       if $|m_{n-1}| < r$ then {return Invalid;}
2. $|Pad(N)| = |Pad(A)| = r$
3. $x = K_r \oplus N$ , $w = K_c$
4. $N_r \| N_c = \pi(x \| w)$
5. $x = N_r \oplus A$, $w = N_c \oplus w$
6. $A_r \| A_c = \pi(x \| w)$
7. $x = A_r$,$w = A_c \oplus w$
8. $a_0 = x, b_0 = w$
9. $c_0 = a_0 \oplus m_0$
10. for $i = 1 \rightarrow n$ do
       $x^{'} \| w^{'} = \pi(c_{i-1} \| b_{i-1})$
       $b_i = b_{i-1} \oplus w^{'}$
       $a_i = x^{'}$
       $c_i = a_i \oplus m_i$
11. $C = c_0 \| c_1 \| ..... \| c_n$
12. $x = a_n, w = b_n$
13. $x = x \oplus K$
14. $K_r^{'} \| K_c^{'} = \pi(x \| w)$
15. $x = K_r^{'} \oplus N$ , $w = K_c^{'} \oplus w$
16. $N_r \| N_c = \pi(x \| w)$
17. $x = N_r \oplus A$, $w = N_c \oplus w$
18. $A_r \| A_c = \pi(x \| w)$
19. $x = A_r$,$w = A_c \oplus w$
20. $z_0 = x$
21. $T = z_0$
22. Return $(C, T)$
```

On Decryption-Query (N, A, C, T)

```
1. $Pad(C) = c_0 \| c_1 \| ..... \| c_n$, Where $|c_i| = r$
2. $|Pad(N)| = |Pad(A)| = r$
3. $x = K_r \oplus N$ , $w = K_c$
4. $N_r \| N_c = \pi(x \| w)$
5. $x = N_r \oplus A$, $w = N_c \oplus w$
6. $A_r \| A_c = \pi(x \| w)$
7. $x = A_r$,$w = A_c \oplus w$
8. $a_0 = x, b_0 = w$
9. $m_0 = a_0 \oplus c_0$
10. for $i = 1 \rightarrow n$ do
       $x^{'} \| w^{'} = \pi(c_{i-1} \| b_{i-1})$
       $b_i = b_{i-1} \oplus w^{'}$
       $a_i = x^{'}$
       $m_i = a_i \oplus c_i$
11. $M = m_0 \| m_1 \| ..... \| m_n$
12. $x = a_n, w = b_n$
13. $x = x \oplus K$
14. $K_r^{'} \| K_c^{'} = \pi(x \| w)$
15. $x = K_r^{'} \oplus N$ , $w = K_c^{'} \oplus w$
16. $N_r \| N_c = \pi(x \| w)$
17. $x = N_r \oplus A$, $w = N_c \oplus w$
18. $A_r \| A_c = \pi(x \| w)$
19. $x = A_r$,$w = A_c \oplus w$
20. $z_0 = x$
21. if $(z_0 == T)$ then
       Return $(a_0, b_0)$
    else
       Return $\perp$
```

On π-Query m, where $m \in \{0,1\}^b$

```
1. let $(x \| w) = m$, where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. if $(m, v) \in I_\pi$ then return $v$
3. $v \xleftarrow{\$} \{0,1\}^b$
4. if $\exists\ m'$ s.t $(m', v) \in I_\pi$, then $v \xleftarrow{\$} \{0,1\}^b \setminus \{v : (*, v) \in I_\pi\}$,
   where $* \in \{0,1\}^b$
5. $I_\pi = I_\pi \bigcup \{(m, v)\}$
6. return $v$;
```

On π^{-1}-Query $v = \{v_1 || v_2\}$. where $v_1 \in \{0,1\}^r,\ v_2 \in \{0,1\}^c,\ v \in \{0,1\}^b$

```
1. if $(m, v) \in I_\pi$ then return $m$
2. $m \xleftarrow{\$} \{0,1\}^b$
3. if $\exists\ v'$ s.t $(m, v') \in I_\pi$, then
   $m \xleftarrow{\$} \{0,1\}^b \setminus \{m : (m, *) \in I_\pi\}$, where $* \in \{0,1\}^b$
4. $I_\pi = I_\pi \bigcup \{(m, v)\}$
5. return $m$;
```

Fig. 23. Game $G0'$

Game $\boxed{G1'}$ **and Game** $G2'$ **:** Initialise $IV_1 = 0^r, IV_2 = 0^c$, $K \xleftarrow{\$} \{0,1\}^r$, $K_r \| K_c \xleftarrow{\$} \{0,1\}^b$, $I_\pi = \{((IV_1 \oplus K)\|IV_2, K_r\|K_c)\}$

On Encryption-Query $(N, A, M, flag)$
Same as Game 0

On Decryption-Query (N, A, C, T)
Same as Game 0

On π**-Query** m,where $m \in \{0,1\}^b$

```
1. let $(x||w)$=$m$,where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. if $(m,v) \in I_\pi$ then return $v$
3. $v \xleftarrow{\$} \{0,1\}^b$
4. if $\exists\, m'$ s.t $(m',v) \in I_\pi$, then bad←true and
   $\boxed{v \xleftarrow{\$} \{0,1\}^b \setminus \{v : (*,v) \in I_\pi\}}$, where $* \in \{0,1\}^b$
5. $I_\pi = I_\pi \bigcup \{(m,v)\}$
6. return $v$;
```

On π^{-1}**-Query** v, where $v \in \{0,1\}^b$

```
1. let $(v_1||v_2)$=$m$,where $v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$,
2. if $(m,v) \in I_\pi$ then return $m$
3. $m \xleftarrow{\$} \{0,1\}^b$
4. if $\exists\, v'$ s.t $(m,v') \in I_\pi$, then bad←true and
   $\boxed{m \xleftarrow{\$} \{0,1\}^b \setminus \{m : (m,*) \in I_\pi\}}$, where $* \in \{0,1\}^b$
5. $I_\pi = I_\pi \bigcup \{(m,v)\}$
6. return $m$;
```

Fig. 24. Game $G1'$ and Game $G2'$

Game $G3'$ and Game $\boxed{G4'}$: Initialise $IV_1 = 0^r, IV_2 = 0^c,\ K \xleftarrow{\$} \{0,1\}^r,\ K_r\|K_c \xleftarrow{\$} \{0,1\}^b,\ I_\pi = \{((IV_1 \oplus K)\|IV_2, K_r\|K_c)\}$, $I_c = \{w\}$

On Encryption-Query $(N, A, M, flag)$

```
1. if $(flag = 1)$ then
       $M = m_0\|m_1\|.....\|m_{n-1}$, Where $|m_i| = r$ and
       $0 \leq i < (n-1)$
       $Pad(M) = m_0\|m_1\|.....\|(m_{n-1}\|10^{r-(|m_{n-1}|+1)})$
   else
       $M = m_0\|m_1\|...\|m_{n-1}$, Where $|m_i| = r$
       if $|m_{n-1}| < r$ then {return Invalid;}
2. $|Pad(N)| = |Pad(A)| = r$
3. $x = IV_1, w = IV_2$
4. $x = K_r \oplus N\ ,w = K_c$
5. if $\exists v\ s.t.(x\|w, v) \in I_\pi$, then
       $N_r\|N_c = v$
   else
       $N_r\|N_c \xleftarrow{\$} \{0,1\}^b$
       $\boxed{\text{if } (N_c \oplus w) \in I_c \text{ then } bad \leftarrow true}$
           $\boxed{N_c \xleftarrow{\$} \{0,1\}^c \setminus \{N_c' : (N_c' \oplus w) \in I_c\}}$
       $I_\pi = I_\pi \cup \{(x\|w, N_r\|N_c)\}$
6. $x = N_r \oplus A\ ,w = N_c \oplus w$
7. $I_c = I_c \cup \{w\}$
8. if $\exists v\ s.t.\ (x\|w, v) \in I_\pi$, then
       $A_r\|A_c = v$
   else
       $A_r\|A_c \xleftarrow{\$} \{0,1\}^b$
       $\boxed{\text{if } (A_c \oplus w) \in I_c \text{ then } bad \leftarrow true}$
           $\boxed{A_c \xleftarrow{\$} \{0,1\}^c \setminus \{A_c' : (A_c' \oplus w) \in I_c\}}$
       $I_\pi = I_\pi \cup \{(x\|w, A_r\|A_c)\}$
9. $x = A_r\ ,w = A_c \oplus w$
10. $I_c = I_c \cup \{w\}$
11. $a_0 = x, b_0 = w$
12. $c_0 = a_0 \oplus m_0$
13. for $i = 1 \rightarrow n$ do
        if $\exists v\ s.t.\ (c_{i-1}\|b_{i-1}, v) \in I_\pi$, then
            $x'\|w' = v$
        else
            $x'\|w' \xleftarrow{\$} \{0,1\}^b$
            $\boxed{\text{if } (b_{i-1} \oplus w') \in I_c \text{ then } bad \leftarrow true}$
            $\boxed{w' \xleftarrow{\$} \{0,1\}^c \setminus \{w'' : (b_{i-1} \oplus w'') \in I_c\}}$
            $I_\pi = I_\pi \cup \{(c_{i-1}\|b_{i-1}, x'\|w')\}$
        $b_i = b_{i-1} \oplus w'$
        $I_c = I_c \cup \{b_i\}$
        $a_i = x'$
        $c_i = a_i \oplus m_i$
14. $C = c_0\|c_1\|.....\|c_n$
15. $x = a_n, w = b_n$
16. $x = x \oplus K$
17. if $\exists v\ s.t.(x\|w, v) \in I_\pi$, then
        $K_r'\|K_c' = v$
    else
        $K_r'\|K_c' \xleftarrow{\$} \{0,1\}^b$
        $\boxed{\text{if } (w \oplus K_c') \in I_c \text{ then } bad \leftarrow true}$
            $\boxed{K_c' \xleftarrow{\$} \{0,1\}^c \setminus \{K_c'' : (w \oplus K_c'') \in I_c\}}$
        $I_\pi = I_\pi \cup \{(x\|w, K_r'\|K_c')\}$
18. $x = K_r' \oplus N\ ,\ w = K_c' \oplus w$
19. $I_c = I_c \cup \{w\}$
20. Repeat step from 5 to 9.
```

On Decryption-Query (N, A, C, T)

```
1. $Pad(C) = c_0\|c_1\|.....\|c_n$, Where $|c_i| = r$
2. $|Pad(N)| = |Pad(A)| = r$
3. $x = K_r \oplus N\ ,\ w = K_c$
4. if $\exists v\ s.t.(x\|w, v) \in I_\pi$, then
       $N_r\|N_c = v$
   else
       $N_r\|N_c \xleftarrow{\$} \{0,1\}^b$
       $\boxed{\text{if } (N_c \oplus w) \in I_c \text{ then } bad \leftarrow true}$
           $\boxed{N_c \xleftarrow{\$} \{0,1\}^c \setminus \{N_c' : (N_c' \oplus w) \in I_c\}}$
       $I_\pi = I_\pi \cup \{(x\|w, N_r\|N_c)\}$
5. $x = N_r \oplus A,\ w = N_c \oplus w$
6. $I_c = I_c \cup \{w\}$
7. if $\exists v\ s.t.\ (x\|w, v) \in I_\pi$, then
       $A_r\|A_c = v$
   else
       $A_r\|A_c \xleftarrow{\$} \{0,1\}^b$
       $\boxed{\text{if } (A_c \oplus w) \in I_c \text{ then } bad \leftarrow true}$
           $\boxed{A_c \xleftarrow{\$} \{0,1\}^c \setminus \{A_c' : (A_c' \oplus w) \in I_c\}}$
       $I_\pi = I_\pi \cup \{(x\|w, A_r\|A_c)\}$
8. $x = A_r,w = A_c \oplus w$
9. $I_c = I_c \cup \{w\}$
10. $a_0 = x, b_0 = w$
11. $m_0 = a_0 \oplus c_0$
12. for $i = 1 \rightarrow n$ do
        if $\exists v\ s.t.\ (c_{i-1}\|b_{i-1}, v) \in I_\pi$, then
            $x'\|w' = v$
        else
            $x'\|w' \xleftarrow{\$} \{0,1\}^b$
            $\boxed{\text{if } (b_{i-1} \oplus w') \in I_c \text{ then } bad \leftarrow true}$
            $\boxed{w' \xleftarrow{\$} \{0,1\}^c \setminus \{w'' : (b_{i-1} \oplus w'') \in I_c\}}$
            $I_\pi = I_\pi \cup \{(c_{i-1}\|b_{i-1}, x'\|w')\}$
        $b_i = b_{i-1} \oplus w'$
        $I_c = I_c \cup \{b_i\}$
        $a_i = x'$
        $m_i = a_i \oplus c_i$
13. $M = m_0\|m_1\|.....\|m_n$
14. $x = a_n, w = b_n$
15. $x = x \oplus K$
16. if $\exists v\ s.t.(x\|w, v) \in I_\pi$, then
        $K_r'\|K_c' = v$
    else
        $K_r'\|K_c' \xleftarrow{\$} \{0,1\}^b$
        $\boxed{\text{if } (w \oplus K_c') \in I_c \text{ then } bad \leftarrow true}$
            $\boxed{K_c' \xleftarrow{\$} \{0,1\}^c \setminus \{K_c'' : (w \oplus K_c'') \in I_c\}}$
        $I_\pi = I_\pi \cup \{(x\|w, K_r'\|K_c')\}$
17. $x = K_r' \oplus N\ ,\ w = K_c' \oplus w$
18. $I_c = I_c \cup \{w\}$
19. Repeat step from 4 to 8.
20. $z_0 = x$
21. if $(z_0 == T)$ then
        Return $(a_0, b_0)$
    else
```

On π**-Query** m, where $m \in \{0,1\}^b$

```
1. let $(x||w) = m$, where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. **if** $\exists$ $\{v_1, v_2, ...v_t\}$ s.t. $(m, v_i) \in I_\pi$ then return
   $v \xleftarrow{\$} \{v_1, v_2, ...v_t\}$
3. **else** $v \xleftarrow{\$} \{0,1\}^b$
4. $I_\pi = I_\pi \bigcup \{(m, v)\}$
5. return $v$;
```

On π^{-1}**-Query** $v = \{v_1||v_2\}$. where $v_1 \in \{0,1\}^r,\ v_2 \in \{0,1\}^c,\ v \in \{0,1\}^b$

```
1. **if** $\exists$ $\{m_1, m_2, ...m_t\}$ s.t. $(m_i, v) \in I_\pi$ then return
   $m \xleftarrow{\$} \{m_1, m_2, ...vm_t\}$
2. **else** $m \xleftarrow{\$} \{0,1\}^b$
3. $I_\pi = I_\pi \bigcup \{(m, v)\}$
4. return $m$;
```

Fig. 26. Game $G3'$ and Game $G4'$