Bachelor Thesis

Radboud University

Lock-based Authenticated Encryption in a Multi-user Setting

Author:
Stijn Vandenput

Supervisors:
Martijn Stam
Bart Mennink

June 8, 2024

Abstract

In the context of symmetric encryption, primitives are usually evaluated in a single-user setting. The security bounds found by these evaluations often degrade in a multi-user setting. To prevent this degradation, Giacon, Kiltz and Poettering propose giving a primitive a new input to distinguish between users within the context of hybrid encryption [1]. In this work, where we refer to this new input as a lock, we investigate how these locks work in a broader context by looking at generic composition of authenticated encryption. We do this by first formalizing authenticated encryption using locks, and then we look at different compositions following insights from Namprempre, Rogaway and Shrimpton [2]. We investigate three generic composition methods using locks, and conclude that all three are secure.

Contents

1 Introduction

To implement public-key encryption, a hybrid paradigm is typically followed: To encrypt a message, an ephemeral key is generated using a randomized key encapsulation mechanism (KEM). This key is then used to encrypt the message using a deterministic data encapsulation mechanism (DEM). Both the KEM and DEM output their own ciphertexts, which are concatenated to form the public-key encryption ciphertext. Classical analysis of DEMs considers a single user. Because of this, the security bounds of these analysis do not always transfer to the real world where one can have millions of users. To make sure the security bounds are also good if there are many users, one can use larger ephemeral keys. Although using larger keys for DEMs is generally safer, expanding the size of the key is not always a viable option. This might be due to limitations in computing power or memory, or due to security primitives having fixed key sizes. Giacon, Kiltz and Poettering, henceforth GKP, propose augmentation using locks (originally called tags, but renamed to locks here to avoid overloaded terms) as an alternative solution to key expansion in a multi-user setting [1]. Augmentation with locks works by giving the security primitive an additional input field, called the lock, to distinguish multiple users using the same key. The augmentation can improve security in a multi-user setting, without the need to expand the key. After defining this augmentation, GKP apply the augmentation to a DEM and a MAC function, to create an augmented DEM (ADEM) and an augmented MAC (AMAC). These two are combined to construct an authenticated encryption primitive following the generic encrypt-then-MAC composition from Bellare and Namprempre [3]. This composition is proven secure whenever the underlying ADEM and AMAC are secure.

The generic composition of authenticated encryption has, since the original study of Bellare and Namprempre, been revised by Namprempre, Rogaway and Shrimpton [2], which we henceforth call NRS. In this revision, the generic composition using a MAC function and a deterministic encryption primitive is more thoroughly investigated. The original study found, only the encrypt-then-mac composition to be secure. NRS shine light on the fact the security of the generic compositions depends on the kind of encryption primitives used. After this, they evaluate all possible generic compositions using an nonce-based authenticated encryption primitive and a MAC function. When using these primitives, three composition methods called encrypt-and-MAC, encrypt-then-MAC and MAC-then-encrypt are proven to be secure whenever the underlying primitives are secure.

Although the ADEM+AMAC composition given by GKP is secure, it is not clearly defined as authenticated encryption primitive. As a result, its security is evaluated as a DEM, instead of as an authenticated encryption primitive. Additionally, only the encrypt-then-MAC composition is considered by GKP and it is not shown if other composition methods could be secure. More thoroughly investigating authenticated encryption using locks will lead to a better understanding of the security of locks, and wether or not it is a viable solution to security degradation in a multi-user setting. Additionally, a more precise specification of the lock-based primitives will be a first step towards using the locks outside the context of hybrid encryption. In this thesis, generic composition of authenticated encryption using locks is more thoroughly investigated. We formally compare the constructions and the notation from NRS and GKP (Chapter 3). To evaluate generic composition using locks, we define a new cryptographic primitive which we analyze in a multi-user setting (Chapter 4). Using the knowledge from NRS, three generic compositions (encrypt-and-MAC, encrypt-then-MAC and MAC-then-encrypt) of this primitive are considered, all using a lock-based encryption primitive and a lock-based MAC function (Chapter 5). As our main result, we prove these three compositions to be secure, whenever the underlying primitives are secure (Chapter 6). Afterwards, some use cases of the new primitive are discussed (Chapter 7).

2 Preliminaries

In this section we will explain several concepts important to the rest of our work, as well as some general notation.

2.1 General Notation

Strings are binary and bit-wise, the set of all strings is $\{0,1\}^*$. The length of x is written as $|x|$, the concatenation of x and y as $x \parallel y$, a being the result of b as $a \leftarrow b$, and taking a uniform random sampling from set z and assigning it to x as $x \overset{\$}{\leftarrow} z$. We write N for the number of users and allow a single type of error message written as $\perp$. Any tuple containing $\perp$ will be $\perp$ as well. We define the following spaces, all of them being subsets of the set of all strings: nonempty key space $\mathcal{K}$, lock space $\mathcal{L}$, nonce space $\mathcal{N}$, message space $\mathcal{M}$, ciphertext space $\mathcal{C}$, tag space $\mathcal{T}$, and associated data space $\mathcal{A}$. Unless stated otherwise, $\mathcal{M}$ contains at least two strings, and if $\mathcal{M}$ or $\mathcal{A}$ contains a string of length x, it must contain all strings of length x. There are no further constraints on these spaces.

2.2 Authenticated Encryption

Two different security requirements are data privacy, the insurance that data cannot be viewed by an unauthorized party, and data integrity, the insurance that data has not been modified by an unauthorized party. Authenticated encryption combines both of these security requirements into one and ensures both data privacy and integrity. A basic authenticated encryption scheme consists of a encryption call and a decryption call. The encryption call takes a message and a key to a self-authenticating ciphertext. The decryption call takes a self-authenticating ciphertext and a key to a message. Some authenticated encryption schemes allow an additional input AD, short for associated data. The associated data is specifically required to not have data privacy but does require data integrity. Authenticated encryption schemes that support AD are called AEAD schemes.

2.3 Message Authentication

Message authentication can be done using a message authentication code, MAC for short. A basic MAC function takes a message and key and outputs a tag which authenticates the message. Some MAC functions also have a verification call that takes a message, key and tag and outputs either *true* or *false*. A MAC function can have different security requirements. It is said to be PRF secure when it is infeasible to distinguish the output tag from the result of a pseudo-random function that takes all message-key pairs to the tag space. A MAC is said to be unforgeable when it is infeasible to create a valid message-tag pair without knowledge of the secret key. A PRF secure MAC is also unforgeable, given the tag space is big enough, while a unforgeable MAC is not necessarily PRF secure.

2.4 Nonces and Locks

A basic deterministic encryption scheme takes a message and key as input, and outputs a ciphertext. Using this encryption scheme, a message encrypted under the same key leads to the same ciphertext. Both GPK (Giacon, Kiltz and Poettering) and NRS (Namprempre, Rogaway and Shrimpton) resolve this by giving the encryption scheme an additional argument. GKP uses locks while NRS uses nonces. Although nonces and locks look similar, their purpose and exact working differ leading to different use cases. Most notably, nonces are useful when one user is allowed to encrypt multiple messages and locks are only useful when there are multiple users.

Nonces Using a basic deterministic encryption scheme, a message encrypted twice by the same user results in the same ciphertext. This can leak information about the message, which can be prevented by using a nonce. A nonce is a number that is assumed to only be used once per user to encrypt a message. Whenever a message is encrypted twice with the same key, but with two different nonces, the resulting ciphertexts should be indistinguishable from two ciphertexts corresponding to two different messages. As a result, it is infeasible for an adversary to guess if a message has been sent multiple times. The adversary is usually allowed to let a user decrypt multiple messages with one nonce. Nonces are only used when a user uses its key multiple times, as otherwise a message will never be encrypted by the same user twice.

Locks Using a basic deterministic encryption scheme, a message encrypted by two users that have the same key results in the same ciphertext. This can leak information about the secret keys used, which can be prevented by using locks. Whereas nonces are bound to the message, locks are bound to the user. Each user has one lock, provided the users have one key each, and will encrypt all their messages using that lock. Whenever a message is encrypted twice with the same key, but with two different locks, the resulting ciphertexts should be indistinguishable from two ciphertexts corresponding to two different messages. As a result, it is infeasible for an adversary to see when two users have a key collision unless locks collide as well. To prevent collisions in locks, we assume locks to be globally unique. The adversary is usually only allowed to let a user decrypt messages with the correct lock. Locks are only used in a multi-user setting, as key collision is impossible when there is only one user.

2.5 Security Notions

The security of a cryptographic construction can be modeled as a distinguishing advantage. When doing this, different security notions are formed based on what one distinguishes on. To understand NRS and GKP and how they differ, it is important to understand which security notions they use, all the relevant notions are written below.

Active of Passive Security can be modeled against an passive or an active attacker. An passive attacker can only read the messages while an active attacker can also alter the messages. An passive attacker can be modelled using a chosen plaintext attack, CPA for short. In this model, the adversary can choose the plaintext that is encrypted, but not the ciphertext that is decrypted. An active attacker can be modelled using a chosen ciphertext attack, CCA for short. In this model, the adversary can choose the plaintext that is encrypted, as well as the ciphertext that is decrypted. Shorthand notations for the two is IND-CPA and IND-CCA, respectively. IND-CCA implies IND-CPA, but not the other way around. Both GKP and NRS model the authenticated encryption primitive using IND-CPA and the underlying encryption primitive using IND-CCA.

$ or Left-or-right Left-or-right-indistinguishablility refers to a situation where the adversary gives two messages, and is given a ciphertext. The adversary has to guess which of the two messages corresponds to the ciphertext. $-indistinguishablility refers to a situation where the adversary is given access to either the real construction, or to a lazily sampled random function \$. This random function returns a random string with the same length as the ciphertext would have. The adversary has to guess which of these two it has access to. As long as the length of the ciphertext only depends on the length of the message, not its content, IND-\$ implies IND-LOR, but not the other way around. IND-\$ is used by GPK and IND-LOR is used by NRS.

Both of these are separate dimensions and they can be combined into 4 different notions. For example IND-\$-CCA refers to a situation where the adversary has to distinguish between the real construction, or a random function while being able to choose both the plaintext that is encrypted and the ciphertext that is decrypted.

Game Based Security Notions These security notions can be written in a game-based format, using pseudocode instead of text. As a example, the IND-\$-CPA game of a nonce-based encryption scheme can be found in Figure 1. A challenge bit b is given to the game, in this case b signals whether we are in the real or the ideal world. The adversary guesses this bit and returns b', signaling its guess for b. In addition, the adversary can have access to oracles. In our example there is only one oracle that takes a nonce and a message. Using game based notation, one can clearly write out all the limitations. For example, the limitation that nonces cannot be reused is modeled by lines 0, 5 and 6. Lines 8 and 9 model how the random function \$ behaves. These limitations could be written out in text based format as well but when there are multiple limitations, writing it out in a game based format can make both the security notion, as well as the security proofs, more comprehensible and precise.

2.6 Security Proofs of Generic Composition

To prove the security of a generic composition we use a security reduction. To define the reduction we bind the advantage of the generic composition by terms of the advantages of the underlying primitives. To do this we show that an advantage on the generic composition can be leveraged to gain an advantage on the underlying primitives. In other words, we prove that if we can break the security of the generic composition, then we can break the security of one of the underlying primitives. After proving this, we can conclude the composition is secure as long as the underlying primitives are secure. The security reduction is said to be tight when its security does not depend on the attack and lose when its security does depend on the adversary's attack.

3 NRS and GKP in Detail

In this section we explain the parts from GKP and NRS important to our work. Afterwards, a comparison is made between the two. Some notations will be different from the original papers for improved consistency. What are called tags by GKP, we will call locks instead to avoid confusion with the output of MAC functions and we call the output of the AMAC the tag instead of the ciphertext. The security notions from NRS are converted to a game-based format using insights from [4] in order to better match the notation from GKP and be more adaptable to a multi-user setting. The security games are only explained briefly in this section, a more in-depth explanation of the relevant constructs can be found in section 4.2.

3.1 NRS

Three generic ways to compose an authenticated encryption scheme are discussed in a paper written by Bellare and Namprempre [3]: encrypt-then-MAC, encrypt-and-MAC and MAC-then-

```
Game nE-IND-$-CPA$^b_A$
    $U \leftarrow \emptyset$
    $k \xleftarrow{\$} \mathcal{K}$
    $b' \leftarrow A$
    return $b'$

Oracle Oenc$(n, m)$
    if $n \in U$ : return $\perp$
    $U \leftarrow U \cup \{n\}$
    $c \leftarrow \mathrm{E}(k, n, m)$
    if $b = 1 \wedge c \neq \perp$ :
        $c \xleftarrow{\$} \{0,1\}^{|c|}$
    return $c$
```

Figure 1: nE-IND-\$-CPA game, A has access to oracle Oenc.

encrypt. In this paper, encrypt-then-MAC is considered the only secure composition when using probabilistic encryption as a building block. NRS note that the type of encryption scheme used influences which compositions are secure, and thus the result of Bellare en Namprempre is only applicable when using probabilistic encryption. Afterwards, the compositions are generalized to using nonce-based encryption, nA for short, and a PRF secure MAC function as a building blocks to create nonce-based authenticated encryption schemes, nAEs for short. With these primitives, all three earlier named compositions are proven secure. Additionally, NRS includes Associated Data (AD) in the authenticated encryption primitive. Below we describe the primitives, their security and the compositions more in depth.

3.1.1 Primitives

nE A nonce-based encryption scheme is defined by a triple $\mathit{\Pi} = (\mathcal{K}, \mathrm{E}, \mathrm{D})$. Deterministic encryption algorithm E takes three inputs (k, n, m) and outputs a value c, the length of c only depends the length of k, n and m. If, and only if, (k, n, m) is not in $\mathcal{K} \times \mathcal{N} \times \mathcal{M}$, c will be $\perp$. Decryption algorithm D takes three inputs (k, n, c) and outputs a value m. Both E and D are required to satisfy correctness (if $\mathrm{E}(k, n, m) = c \neq \perp$, then $\mathrm{D}(k, n, c) = m$) and tidiness (if $\mathrm{D}(k, n, c) = m \neq \perp$, then $\mathrm{E}(k, n, m) = c$).

nE security The security of a nE is defined as

$$\mathbf{Adv}^{\mathrm{nE}}_{\mathit{\Pi},A} = \Pr[\text{nE-IND-\$-CPA}^0_A = 0] - \Pr[\text{nE-IND-\$-CPA}^1_A = 0]$$

where nE-IND-\$-CPA is in Figure 1. Set U keeps track of all used nonces as the adversary is not allowed to repeat nonces.

MAC A MAC is defined by an algorithm F that takes a key k in $\mathcal{K}$ and a string m and outputs either a n-bit tag t or $\perp$. The domain of F is the set X off al m such that $\mathrm{F}(k, m) \neq \perp$ is in X, this domain may not depend on k.

MAC security NRS require the MAC to be PRF secure. The security is defined as

$$\mathbf{Adv}^{\mathrm{MAC}}_{\mathrm{F},A} = \Pr[\text{MAC-PRF}^0_A = 0] - \Pr[\text{MAC-PRF}^1_A = 0]$$

where MAC-PRF is in Figure 2. In this game the set U keeps track of the used messages to prevent trivial distinctions.

```
Game MAC-PRF$^b_A$
    $U \leftarrow \emptyset$
    $k \xleftarrow{\$} \mathcal{K}$
    $b' \leftarrow A$
    return $b'$

Oracle Omac$(m)$
    if $m \in U$ : return $\perp$
    $U \leftarrow U \cup \{m\}$
    $t \leftarrow \mathrm{F}(k, m)$
    if $b = 1 \wedge t \neq \perp$ :
        $t \xleftarrow{\$} \{0,1\}^{|t|}$
    return $t$
```

Figure 2: MAC-PRF, A has access to oracle Omac and U is the set of used messages.

```
Game nAE-IND-\$-CCA$^b_A$
    $U \leftarrow \emptyset$
    $Q \leftarrow \emptyset$
    $k \xleftarrow{\$} \mathcal{K}$
    $b' \leftarrow A$
    return $b'$

Oracle Oenc$(n, a, m)$
    if $n \in U$ : return $\perp$
    $U \leftarrow U \cup \{n\}$
    if $(n, a, m, \_) \in Q$ : return $\perp$
    $c \leftarrow \mathrm{E}(k, n, a, m)$
    if $b = 1 \wedge c \neq \perp$ :
        $c \xleftarrow{\$} \{0,1\}^{|c|}$
    $Q \leftarrow Q \cup \{(n, a, m, c)\}$
    return $c$

Oracle Odec$(n, a, c)$
    if $b = 1$ : return $\perp$
    if $(n, a, \_, c) \in Q$ : return $\perp$
    $m \leftarrow \mathrm{D}(k, n, a, c)$
    $Q \leftarrow Q \cup \{(n, a, m, c)\}$
    return $m$
```

Figure 3: nAE-IND-$-CCA game, A has access to oracles Oenc and Odec.

nAE A nonce-based authenticated encryption scheme is defined by a triple $\mathit{\Pi} = (\mathcal{K}, \mathrm{E}, \mathrm{D})$. Deterministic encryption algorithm E takes four inputs (k, n, a, m) and outputs a value c, the length of c only depends the length of k, n, a and m. If, and only if, (k, n, a, m) is not in $\mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$, c will be $\perp$. Decryption algorithm D takes four inputs (k, n, a, c) and outputs a value m. both E and D are required to satisfy correctness (if $\mathrm{E}(k, n, a, m) = c \neq \perp$, then $\mathrm{D}(k, n, a, c) = m$) and tidiness (if $\mathrm{D}(k, n, a, c) = m \neq \perp$, then $\mathrm{E}(k, n, a, m) = c$).

nAE security The security of a nAE is defined as

$$\mathbf{Adv}^{\mathrm{nAE}}_{\mathit{\Pi},A} = \Pr[\text{nAE-IND-\$-CCA}^0_A = 0] - \Pr[\text{nAE-IND-\$-CCA}^1_A = 0]$$

where nAE-IND-$-CCA is in Figure 3. The adversary is not allowed to repeat nonces on encryption, and set U keeps track of all used nonces. Following the translation of IND-$-CCA to a security game for AE from [4], _ denotes a variable that is irrelevant and set Q keeps tack of all query results in order to prevent trivial distinctions.

3.1.2 Composition

NRS define 20 different schemes that compose an nAE from an nE and a MAC function. They define a composition to be secure if there is a tight reduction from breaking the nAE-security of the scheme to breaking the nE-security and the PRF security of the underlying primitives. Three different schemes, named N1, N2 and N3, were proven to be secure. Noteworthy is that these relate to encrypt-and-MAC, encrypt-then-MAC and MAC-then-encrypt, respectively.

```
Game L-IND-LOR-CPA$^b_{A,N}$
0 :  $L \leftarrow \emptyset$
1 :  for $j \in [1..N]$ :
2 :      $k_j \xleftarrow{\$} \mathcal{K}$
3 :      $C_j \leftarrow \emptyset$
4 :  $b' \leftarrow A$
5 :  return $b'$

Oracle Oenc$(j, l, m_0, m_1)$
6 :  if $C_j \neq \emptyset$ : return $\bot$
7 :  if $l \in L$ : return $\bot$
8 :  $L \leftarrow L \cup \{l\}$
9 :  $l_j \leftarrow l$
10 : $c \leftarrow$ A.enc$(k_j, l_j, m_b)$
11 : $C_j \leftarrow C_j \cup \{c\}$
12 : return $c$
```

Figure 4: L-IND-LOR-CPA game, A has access to oracle Oenc.

Additionally, they propose a scheme N4, for which they can not prove it secure, nor find a counterexample to prove it is insecure. However, this case has since been proven to be insecure as well [5] so is not considered here. All four schemes can be viewed in Figure 6 of the original paper by NRS.

3.2 GKP

GKP discusses the concept of augmentation using locks to prevent security degradation in a multi-user setting. They start by showing how some data encapsulation mechanisms are vulnerable to both a passive multi-instance distinguishing attack and key recovery attack. These vulnerabilities lead to the degradation of the security bounds found in analyses considering only a single user. As an alternative solution to expanding the key size, they define the augmented data encapsulation mechanisms, ADEM for short, that uses locks to prevent this degradation. Additionally, they show how an ADEM that is secure against passive attacks can be combined with a MAC that is augmented in a similar fashion, called an AMAC, to construct an ADEM that is safe against active attackers. Below we describe the primitives, their security and the composition more in depth.

3.2.1 Primitives

ADEM An ADEM scheme is defined by a tuple (A.enc, A.dec). Deterministic algorithm A.enc takes a key k in $\mathcal{K}$, a lock l in $\mathcal{L}$ and a message m in $\mathcal{M}$ and outputs a ciphertext c in $\mathcal{C}$. Deterministic algorithm A.dec takes a k in $\mathcal{K}$, a lock l in $\mathcal{L}$ and a ciphertext c in $\mathcal{C}$ and outputs a message m in $\mathcal{M}$ or $\bot$ to indicate rejection. The correctness requirement is that for every combination of k, l and m we have A.dec$(k, l,$ A.enc$(k, l, m)) = m$. We will consider both CPA and CCA security separately for this scheme.

ADEM-CPA security The security of an ADEM-CPA, just called ADEM by GKP, is defined as

$$\mathbf{Adv}^{\text{l-ind-lor-cpa}}_{\text{ADEM},A,N} = \Pr[\text{L-IND-LOR-CPA}^0_{A,N} = 0] - \Pr[\text{L-IND-LOR-CPA}^1_{A,N} = 0]$$

where L-IND-LOR-CPA is in Figure 4. Every user is only allowed one encryption as enforced by lines 3, 6 and 11. Locks may not repeat between users as enforced by lines 0, 7, 8 and 9. The corresponding game can be found in Figure 9 from GKP. Note that this figure also includes a decryption oracle, but the adversary is not allowed to use this oracle considering CPA security.

```
**Game** L-IND-LOR-CCA$^b_{A,N}$
0 : $L \leftarrow \emptyset$
1 : **for** $j \in [1..N]$ :
2 :     $k_j \xleftarrow{\$} \mathcal{K}$
3 :     $C_j \leftarrow \emptyset$
4 : $b' \leftarrow A$
5 : **return** $b'$

**Oracle** Oenc$(j, l, m_0, m_1)$
6 : **if** $C_j \neq \emptyset$ : **return** $\bot$
7 : **if** $l \in L$ : **return** $\bot$
8 : $L \leftarrow L \cup \{l\}$
9 : $l_j \leftarrow l$
10 : $c \leftarrow$ A.enc'$(k_j, l_j, m_b)$
11 : $C_j \leftarrow C_j \cup \{c\}$
12 : **return** $c$

**Oracle** Odec$(j,c)$
13 : **if** $C_j = \emptyset$ : **return** $\bot$
14 : **if** $c \in C_j$ : **return** $\bot$
15 : $m \leftarrow$ A.dec'$(k_j, l_j, c)$
16 : **return** $m$
```

Figure 5: L-IND-LOR-CCA game, A has access to oracles Oenc and Odec and the locks in line 10 and 15 are the same.

ADEM-CCA security The security of an ADEM-CCA, called ADEM' by GKP, is defined as

$$\mathbf{Adv}^{\text{l-ind-lor-cca}}_{\text{ADEM'},A,N} = \Pr[\text{L-IND-LOR-CCA}^0_{A,N} = 0] - \Pr[\text{L-IND-LOR-CCA}^1_{A,N} = 0]$$

where L-IND-LOR-CCA is in Figure 5. Every user is only allowed one encryption query as enforced by lines 3, 6 and 11. Locks may not repeat between users as enforced by lines 0, 7, 8 and 9. Decryption queries are only allowed after the given user made an encryption as enforced by lines 3, 11 and 13. Line 14 prevents trivial distinctions. The corresponding game can be found in Figure 9 of GKP.

AMAC An AMAC scheme is defined by a tuple (M.mac, M.vrf). Deterministic algorithm M.mac takes a key k in $\mathcal{K}$, a lock l in $\mathcal{L}$, and a message m in $\mathcal{M}$ and outputs a tag t in $\mathcal{T}$. Deterministic algorithm M.vrf takes a key k in $\mathcal{K}$, a lock l in $\mathcal{L}$, a message m in $\mathcal{M}$ and a tag t in $\mathcal{T}$ and returns either *true* or *false*. The correctness requirement is that for every combination of k, l and m, all corresponding $t \leftarrow$M.mac(k, l, m) gives M.vrf(k, l, m, t) = *true*.

AMAC security The security of a AMAC is defined as

$$\mathbf{Adv}^{\text{L-MIOT-UF}}_{\text{AMAC},A,N} = \Pr[\text{L-MIOT-UF}_{A,N} = 1]$$

where L-MIOT-UF is in Figure 6. Every user is only allowed one MAC query as enforced by lines 4, 7 and 12. Locks may not repeat between users as enforced by lines 1, 8, 9 and 10. Verification queries are only allowed after the user made an mac query as enforced by lines 4, 12 and 14. Line 15 prevents trivial distinctions. The corresponding game can be found in Figure 15 of GKP.

Notational Differences GKP do not require $\mathcal{M}$ to contain at least two strings, and to contain all strings of length x if it contains a string of length x. Additionally, $\mathcal{K}$ is required to be finite but not required to be non-empty.

3.2.2 Composition

GKP construct an ADEM scheme that is CCA secure, using an ADEM scheme that is CPA secure and an AMAC scheme. The composition follows the the encrypt-then-MAC method from Bellare and Namprempre [3] and is thus similar to composition N2 from NRS. The resulting

```
Game L-MIOT-UF$_{A,N}$
0 : forged $\leftarrow 0$
1 : $L \leftarrow \emptyset$
2 : for $j \in [1..N]$ :
3 :     $k_j \xleftarrow{\$} \mathcal{K}$
4 :     $T_j \leftarrow \emptyset$
5 : run $A$
6 : return forged

Oracle Omac$(j, l, m)$
7 : if $T_j \neq \emptyset$ : return $\perp$
8 : if $l \in L$ : return $\perp$
9 : $L \leftarrow L \cup \{l\}$
10 : $l_j \leftarrow l$
11 : $t \leftarrow$ M.mac$(k_j, l_j, m)$
12 : $T_j \leftarrow T_j \cup \{(m, t)\}$
13 : return $t$

Oracle Ovrf$(j, m, t)$
14 : if $T_j = \emptyset$ : return $\perp$
15 : if $(m, t) \in T_j$ : return $\perp$
16 : if M.vrf$(k_j, l_j, m, t)$ :
17 :     forged $\leftarrow 1$
18 :     return $true$
19 : else : return $false$
```

Figure 6: L-MIOT-UF game, A has access to oracles Omac and Ovrf and the locks in line 11 and 16 are the same.

```
Proc A.enc'$(k, l, m)$
$(k_{dem}, k_{mac}) \leftarrow k$
$c' \leftarrow$ A.enc$(k_{dem}, l, m)$
$t \leftarrow$ M.mac$(k_{mac}, l, c')$
$c \leftarrow (c', t)$
return $c$

Proc A.dec'$(k, l, c)$
$(k_{dem}, k_{mac}) \leftarrow k$
$(c', t) \leftarrow c$
if M.vrf$(k_{mac}, l, c', t)$ :
    $m \leftarrow$ A.dec$(k_{dem}, l, c')$
    return $m$
else : return $\perp$
```

Figure 7: A.enc and A.dec calls, The corresponding calls can be found in Figure 16 of GKP.

algorithms A.enc' and A.dec' are in Figure 7. They define the composition to be secure as there is a tight reduction from breaking the ADEM-CCA of the scheme to breaking the ADEM-CPA or the AMAC security of the underlying primitives.

3.3 Comparison of GKP and NRS

In this section we will highlight how GKP and NRS differ, as well as why.

Context and Aim Historically, a single user that reuses a single key is considered in a symmetric context, NRS follows this trend as they wrote in this context. In contrast, GKP wrote in the context of hybrid encryption, a context that considers multiple users that use their encryption key once. Apart from this difference in context, there is also a different aim. While NRS aims to generalize the generic nAE composition, GKP aims to find a single composition that can be used for hybrid encryption. Most notably, this results in NRS evaluating 20 possible compositions while GKP evaluates one. Additionally, NRS incorporates AD while GKP does not.

Security Notion The security notions from both papers also reflect the differences in contexts. NRS writes the security notions in a IND-$ fashion, common in symmetric cryptography. Conversely, GKP writes them in a IND-LOR fashion, common in Hybrid encryption. In other words, NRS requires the valid ciphertext to be indistinguishable from random strings while GKP requires them to be indistinguishable from each other. As a result, the MAC primitives of the

```
Game 1AE-IND-$-CCA$^b_{A,N}$
    $L \leftarrow \emptyset$
    for $j \in [1..N]$ :
        $k_j \xleftarrow{\$} \mathcal{K}$
        $C_j \leftarrow \bot$
    $b' \leftarrow A$
    return $b'$

Oracle Oenc$(j, l, m)$
    if $C_j \neq \bot$ : return $\bot$
    if $l \in L$ : return $\bot$
    $L \leftarrow L \cup \{l\}$
    $l_j \leftarrow l$
    $c \leftarrow$ AE.enc$(k_j, l_j, m)$
    if $b = 1 \wedge c \neq \bot$ :
        $c \xleftarrow{\$} \{0,1\}^{|c|}$
    $C_j \leftarrow c$
    return $c$

Oracle Odec$(j, c)$
    if $C_j = \bot$ : return $\bot$
    if $c = C_j$ : return $\bot$
    $m \leftarrow$ AE.dec$(k_j, l_j, c)$
    if $b = 1$ : $m \leftarrow \bot$
    return $m$
```

Figure 8: lAE-IND-$-CCA game, adversary has access to oracles Oenc and Odec.

two papers have different security requirements. NRS requires the tag to be indistinguishable from a random string while GKP requires the tag to be unforgeable. Furthermore, NRS considers nonces while GKP considers locks to match their respective settings.

4 Lock-based Authenticated Encryption

To evaluate the security of generic composition using locks, we define a new security primitive: the lock-based Authenticated Encryption scheme, lAE scheme for short. This lAE is similar to a the nAE from NRS, but it uses locks instead of nonces. Additionally it does not use associated data (AD). We will evaluate the security in a multi user setting where encryption keys are used once.

4.1 lAE

A lAE scheme is defined by a tuple (AE.enc, AE.dec). Deterministic algorithm AE.enc takes three inputs (k, l, m) and outputs a value c, where the length of c only depends on the length of k, l and m. If, and only if, (k, l, m) is not in $\mathcal{K} \times \mathcal{L} \times \mathcal{M}$, c will be $\bot$. Deterministic algorithm AE.dec takes three inputs (k, l, c) and outputs a value m. Both AE.enc and EA.dec are required to satisfy correctness (if AE.enc$(k, l, m) = c \neq \bot$, then AE.dec$(k, l, c) = m$) and tidiness (if AE.dec$(k, l, c) = m \neq \bot$, then AE.enc$(k, l, m) = c$).

4.2 lAE Security Model

The security is defined as

$$\mathbf{Adv}^{\text{lAE}}_{A,N} = \Pr[\text{lAE-IND-\$-CCA}^0_{A,N} = 0] - \Pr[\text{lAE-IND-\$-CCA}^1_{A,N} = 0]$$

where lAE-IND-$-CCA is in Figure 8. Because we consider multiple users who use their keys once, decryption queries of a user are only allowed after an encryption has been made and the user is only allowed one encryption query. On decryption, we use a function that always returns $\bot$ to ensure the adversary cannot guess which ciphertexts would be valid ciphertexts. The idea behind the resulting security game is explained below.

Multiple users Line 1 loops over all the users to initialize with a random key in line 2 and an invalid ciphertext in line 3. Whenever the adversary calls one of the oracles Oenc or Odec, it has to specify user j.

Locks Line 0 initializes the set of all used locks to the empty set. Locks are not allowed to repeat, if the lock is in the set of used locks we return $\perp$ on line 7. If this check passes, we add the lock to the sets of used locks in line 8 and bind it to the user in line 9. Note that locks may be added to the set of used locks even if they are never used to encrypt a valid message. (**todo: see if this needs to be altered**)

One-time use keys The variable C_j is used to prevent multiple encryptions per user. In contrast to GKP, we do not use set notation, as we can never have multiple ciphertexts related to one user. In line 3, we set C_j to be undefined, if the ciphertext is defined in line 6, we return $\perp$. In line 13, the newly computed ciphertext is bound to C_j. If the encryption was invalid, C_j will stay undefined. This leads to the adversary being able to call Oenc twice on a single user, but will not give the adversary an advantage as the values for which AE.enc returns $\perp$ are known. If the user has made no valid encryption yet, decryption is not allowed and we return $\perp$ on line 15 as C_j will be undefined.

Preventing trivial distinctions Line 16 prevents trivial distinctions. If the ciphertext given to Odec is allowed to be the same as the ciphertext returned by Oenc, it would be trivial to distinguish the real and ideal world. In this case, the ideal world would return $\perp$ while the real world would not. For this reason the real world should return $\perp$ as well.

Encryption and decryption If the given arguments are valid, and we are in the real world, line 10 encrypts the message and line 17 decrypts the message.

Implementation of \$ On encryption, whenever AE returns $\perp$, the random function should return $\perp$ as well. Therefore, the random function is only called if $b = 1$ and AE.enc does not return $\perp$. This is checked in line 11. If the check passes, the random function lazily samples a string uniformly at random from the set of all strings with the length of the ciphertext. This random string is bound to the ciphertext in line 12. On decryption, the ideal world always returns $\perp$. **(todo: add part about ideal vs attainable)**

5 Composition

In this section we discuss how we can construct a safe lAE. Similarly to GKP and NRS we will look at compositions combining a deterministic encryption primitive and MAC primitive. First, we write down the definitions of these two primitives, then we will look at how we can combine the two and which security bounds we can expect. Lastly we compare our choices with existing alternatives.

5.1 Used Primitives

lE A lock-based encryption scheme, lE for short, is defined by a tuple (E.enc, E.dec). Deterministic algorithm E.enc takes three inputs (k, l, m) and outputs a value c, the length of c only depends on the length of k, l and m. If, and only if, (k, l, m) is not in $\mathcal{K} \times \mathcal{L} \times \mathcal{M}$, c will be $\perp$. Deterministic algorithm E.dec takes three inputs (k, l, c) and outputs a value m. Both E.enc

```
Game lE-IND-$-CPA$^b_{A,N}$
    $L \leftarrow \emptyset$
    for $j \in [1..N]$ :
        $k_j \xleftarrow{\$} \mathcal{K}$
        $C_j \leftarrow \bot$
    $b' \leftarrow A$
    return $b'$
```

```
Oracle Oenc$(j, l, m)$
    if $C_j \neq \bot$ : return $\bot$
    if $l \in L$ : return $\bot$
    $L \leftarrow L \cup \{l\}$
    $l_j \leftarrow l$
    $c \leftarrow$ E.enc$(k_j, l_j, m)$
    if $b = 1 \wedge c \neq \bot$ :
        $c \xleftarrow{\$} \{0,1\}^{|c|}$
    $C_j \leftarrow c$
    return $c$
```

Figure 9: lE-IND-$-CPA game, Ahas access to oracle Oenc.

and E.dec are required to satisfy correctness (if E.enc$(k, l, m) = c \neq \bot$, then E.dec$(k, l, c) = m$) and tidiness (if E.dec$(k, l, c) = m \neq \bot$, then E.enc$(k, l, m) = c$). Ciphertext space $\mathcal{C}$ consists of all valid ciphertexts.

lE security The security of a lE is defined as

$$\mathbf{Adv}^{\mathrm{lE}}_{A,N} = \Pr[\text{lE-IND-\$-CPA}^0_{A,N} = 0] - \Pr[\text{lE-IND-\$-CPA}^1_{A,N} = 0]$$

where lE-IND-$-CPA is in Figure 9. The user is only allowed one encryption query and decryption queries are only allowed after the encryption. Locks may not repeat between users.

lMAC A lock-based MAC is defined by a deterministic algorithm M.mac that takes a fixed length k in $\mathcal{K}$, a fixed length l in $\mathcal{L}$ and a variable length message m in $\mathcal{M}$ and outputs either a n-bit length string we call tag t, or $\bot$. If, and only if, (k, l, m) is not in $\mathcal{K} \times \mathcal{L} \times \mathcal{M}$, t will be $\bot$. Tag space $\mathcal{T}$ consists of all valid tags.

lMAC security The security of a lock bases, one-time use PRF secure MAC is defined as

$$\mathbf{Adv}^{\mathrm{lMAC}}_{\mathrm{F},A,N} = \Pr[\text{lMAC-PRF}^0_{A,N} = 0] - \Pr[\text{lMAC-PRF}^1_{A,N} = 0]$$

where lMAC-PRF is in Figure 10. Every user is only allowed one MAC query and verification queries are only allowed after the MAC query. Locks may not repeat between users. In contrast to the MAC-PRF from NRS, a verification oracle is needed as we only allow one Omac query per user. In the real world Ovrf will check similar constraints as the Odec from Figure 8. If a Omac query has been made for the given user, and the given message-tag pair is not the result of this query, then the pair is verified. In the ideal world, uniformly random function *tag* is used instead of the lMAC. To define this function we write *Func*$(\mathcal{K} \times \mathcal{L} \times \mathcal{M}, \mathcal{T})$ to denote the set of all functions from key space $\mathcal{K}$, lock space $\mathcal{L}$ and message space $\mathcal{M}$ to tag space $\mathcal{T}$. We need to define this function specifically as we want the tags resulting from computations in oracle Ovrf to match with those in oracle Omac. When the input of *tag* is outside its domain, it will return $\bot$.

```
Game lMAC-PRF$^b_{A,N}$
$L \leftarrow \emptyset$
if $b = 1$ :
    $tag \xleftarrow{\$} Func(\mathcal{K} \times \mathcal{L} \times \mathcal{M}, \mathcal{T})$
for $j \in [1..N]$ :
    $k_j \xleftarrow{\$} \mathcal{K}$
    $T_j \leftarrow \perp$
$b' \leftarrow A$
return $b'$
```

```
Oracle Omac$(j, l, m)$
if $T_j \neq \perp$ : return $\perp$
if $l \in L$ : return $\perp$
$L \leftarrow L \cup \{l\}$
$l_j \leftarrow l$
$t \leftarrow$ M.mac$(k_j, l_j, m)$
if $b = 1 \wedge t \neq \perp$ :
    $t \leftarrow tag(k_j, l_j, m)$
$T_j \leftarrow (m, t)$
return $t$
```

```
Oracle Ovrf$(j, m, t)$
if $T_j = \perp$ : return $\perp$
if $(m, t) = T_j$ : return $\perp$
$t' \leftarrow$ M.mac$(k_j, l_j, m)$
if $b = 1$ :
    $t' \leftarrow tag(k_j, l_j, m)$
if $t = t'$
    return $true$
return $false$
```

Figure 10: lMAC-PRF game, A has access to oracle Omac.

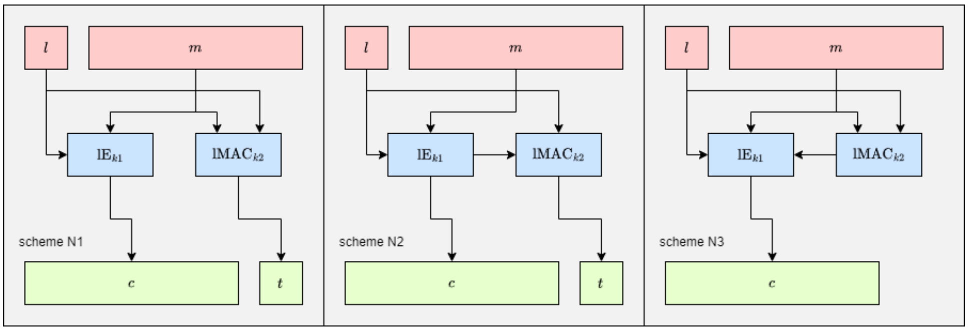

Figure 11: Adjusted N schemes from NRS

5.2 Composition

Following NRS, three ways to construct this lAE are of interest, namely the ones following from the N1, N2 and N3 scheme. The schemes, adjusted to our setting, are in Figure 11. NRS considers 17 more schemes but as none of them has proven to be secure we will not consider those. The AE.enc and AE.dec calls corresponding to N1, N2 and N3 are in Figure 12, 13 and 14 respectively.

5.3 Security Bounds

We define the composition secure if there is a tight reduction from breaking the lAE-security of the scheme to breaking the lE-security or the lMAC security of the underlying primitives. More specifically, we prove the following theorem:

Theorem 1. *Let lAE be constructed from lMAC and lE as described in Figure 12, 13 or 14. Let ciphertext space $\mathcal{C}$ from the lE be a subset of message space $\mathcal{M}$ from the lMAC and let lMAC and lE have a shared lock space. Then, for any number of users N and any lAE adversary A that poses at most Q_e many Oenc queries, and at most Q_d many Odec queries, there exist a lMAC adversary B and a lE adversary C such that:*

$$\mathbf{Adv}^{\text{lAE}}_{A,N} \leq \mathbf{Adv}^{\text{lMAC}}_{B,N} + \mathbf{Adv}^{\text{lE}}_{C,N} + \frac{Q_d}{2^{\text{n}}},$$

```
$\text{AE.enc}(k, l, m)$
    $(k1, k2) \leftarrow k$
    $c' \leftarrow \text{E.enc}(k1, l, m)$
    $t \leftarrow \text{M.mac}(k2, l, m)$
    $c \leftarrow (c', t)$
    return $c$
```

```
$\text{AE.dec}(k, l, c)$
    $(k1, k2) \leftarrow k$
    $(c', t) \leftarrow c$
    $m \leftarrow \text{E.dec}(k1, l, c')$
    $t' \leftarrow \text{M.mac}(k2, l, m)$
    if $t \neq t' : m \leftarrow \perp$
    return $m$
```

Figure 12: AE.enc an AE.dec based on N1

```
$\text{AE.enc}(k, l, m)$
    $(k1, k2) \leftarrow k$
    $c' \leftarrow \text{E.enc}(k1, l, m)$
    $t \leftarrow \text{M.mac}(k2, l, c')$
    $c \leftarrow (c', t)$
    return $c$
```

```
$\text{AE.dec}(k, l, c)$
    $(k1, k2) \leftarrow k$
    $(c', t) \leftarrow c$
    $m \leftarrow \text{E.dec}(k1, l, c')$
    $t' \leftarrow \text{M.mac}(k2, l, c')$
    if $t \neq t' : m \leftarrow \perp$
    return $m$
```

Figure 13: AE.enc an AE.dec based on N2

```
$\text{AE.enc}(k, l, m)$
    $(k1, k2) \leftarrow k$
    $t \leftarrow \text{M.mac}(k2, l, m)$
    $m' \leftarrow m \| t$
    $c \leftarrow \text{E.enc}(k1, l, m')$
    return $c$
```

```
$\text{AE.dec}(k, l, c)$
    $(k1, k2) \leftarrow k$
    $m' \leftarrow \text{E.dec}(k1, l, c)$
    $(m, t) \leftarrow m'$
    $t' \leftarrow \text{M.mac}(k2, l, m)$
    if $t \neq t' : m \leftarrow \perp$
    return $m$
```

Figure 14: AE.enc an AE.dec based on N3

where n *is the output length of the lMAC in bits. The running time of B is at most that of A plus the time required to run Q_e many* E.enc *encapsulations and Q_d many* E.dec *decapsulations. The running time of C is at most that of A. Additionally, B makes at most Q_e many Omac queries and at most Q_d many Ovrf queries and C makes at most Q_e many Oenc queries.*

Within this theorem, both Q_e and Q_d refer to the total queries the adversary is allowed to make, not the queries per user. As a result, Q_e is limited by N.

6 Security proof

To prove theorem 1, we it separately for N1, N2 and N3. In this section, the full proof of case N1 can be found, as well as the main differences between the three cases. The full proof of cases N2 and N3 can be found in appendix A.

N1 First, we repeat theorem 1 specifically for N1:

Theorem 2. *Let lAE be constructed from lMAC and lE as described in Figure 12. Let ciphertext space $\mathcal{C}$ from the lE be a subset of message space $\mathcal{M}$ from the lMAC and let lMAC and lE have a shared lock space. Then, for any number of users N and any lAE adversary A that poses at most Q_e many Oenc queries, and at most Q_d many Odec queries, there exist a lMAC adversary B and a lE adversary C such that:*

$$\mathbf{Adv}^{\mathrm{lAE}}_{A,N} \leq \mathbf{Adv}^{\mathrm{lMAC}}_{B,N} + \mathbf{Adv}^{\mathrm{lE}}_{C,N} + \frac{Q_d}{2^{\mathrm{n}}},$$

where n *is the output length of the lMAC in bits. The running time of B is at most that of A plus the time required to run Q_e many* E.enc *encapsulations and Q_d many* E.dec *decapsulations. The running time of C is at most that of A. Additionally, B makes at most Q_e many Omac queries and at most Q_d many Ovrf queries and C makes at most Q_e many Oenc queries.*

Within this theorem, both Q_e and Q_d refer to the total queries the adversary is allowed to make, not the queries per user. As a result, Q_e is limited by N.

Proof. To prove this theorem, we start by defining game lAE-N1 in Figure 15. This game is the game lAE-IND-\$-CCA (Figure 8), with AE.enc and AE.dec substituted with the N1 algorithms from Figure 12.

```
Game $\text{lAE-N1}^b_{A,N}$
0 :  $L \leftarrow \emptyset$
1 :  for $j \in [1..N]$ :
2 :      $k_j \xleftarrow{\$} \mathcal{K}$
3 :      $C_j \leftarrow \bot$
4 :  $b' \leftarrow A$
5 :  return $b'$
```

```
Oracle $\text{Oenc}(j, l, m)$
6 :  if $C_j \neq \bot$ : return $\bot$
7 :  if $l \in L$ : return $\bot$
8 :  $L \leftarrow L \cup \{l\}$
9 :  $l_j \leftarrow l$
10 : $(k1, k2) \leftarrow k_j$
11 : $c' \leftarrow \text{E.enc}(k1, l_j, m)$
12 : $t \leftarrow \text{M.mac}(k2, l_j, m)$
13 : $c \leftarrow (c', t)$
14 : if $b = 1 \wedge c \neq \bot$ :
15 :     $c \xleftarrow{\$} \{0,1\}^{|c|}$
16 : $C_j \leftarrow c$
17 : return $c$
```

```
Oracle $\text{Odec}(j, c)$
18 : if $C_j = \bot$ : return $\bot$
19 : if $c = C_j$ : return $\bot$
20 : $(k1, k2) \leftarrow k_j$
21 : $(c', t) \leftarrow c$
22 : $m \leftarrow \text{E.dec}(k1, l_j, c')$
23 : $t' \leftarrow \text{M.mac}(k2, l_j, m)$
24 : if $t \neq t'$ : $m \leftarrow \bot$
25 : if $b = 1$ : $m \leftarrow \bot$
26 : return $m$
```

Figure 15: lAE-N1 game, adversary has access to oracles Oenc and Odec.

By definition, this gives us

$$\mathbf{Adv}^{\text{lAE}}_{A,N} = \Pr[\text{lAE-N1}^0_{A,N} = 0] - \Pr[\text{lAE-N1}^1_{A,N} = 0].$$

Next we define game N1-switch-1 in Figure 16. The only difference between this game and game lAE-N1^0 is the fact that N1-switch-1 uses the uniformly random function *tag*, instead of the lMAC. To define this function we write $\mathit{Func}(\mathcal{K} \times \mathcal{L} \times \mathcal{M}, \mathcal{T})$ to denote the set of all functions from the key space of the MAC $\mathcal{K}$, the shared lock space $\mathcal{L}$ and message space $\mathcal{M}$ to the tag space $\mathcal{T}$. We define this function specifically as we want the tags resulting from computations in oracle Oenc to match with those in oracle Odec. When the input of *tag* is outside its domain, it will return $\bot$.

```
Game $\text{N1-switch-1}_{A,N}$
0 :  $L \leftarrow \emptyset$
1 :  $tag \xleftarrow{\$} \mathit{Func}(\mathcal{K}_{mac} \times \mathcal{L} \times \mathcal{M}, \mathcal{T})$
2 :  for $j \in [1..N]$ :
3 :      $k_j \xleftarrow{\$} \mathcal{K}$
4 :      $C_j \leftarrow \bot$
5 :  $b' \leftarrow A$
6 :  return $b'$
```

```
Oracle $\text{Oenc}(j, l, m)$
7 :  if $C_j \neq \bot$ : return $\bot$
8 :  if $l \in L$ : return $\bot$
9 :  $L \leftarrow L \cup \{l\}$
10 : $l_j \leftarrow l$
11 : $(k1, k2) \leftarrow k_j$
12 : $c' \leftarrow \text{E.enc}(k1, l_j, m)$
13 : $t \leftarrow tag(k2, l_j, m)$
14 : $c \leftarrow (c', t)$
15 : $C_j \leftarrow c$
16 : return $c$
```

```
Oracle $\text{Odec}(j, c)$
17 : if $C_j = \bot$ : return $\bot$
18 : if $c = C_j$ : return $\bot$
19 : $(k1, k2) \leftarrow k_j$
20 : $(c', t) \leftarrow c$
21 : $m \leftarrow \text{E.dec}(k1, l_j, c')$
22 : $t' \leftarrow tag(k2, l_j, m)$
23 : if $t \neq t'$ : $m \leftarrow \bot$
24 : return $m$
```

Figure 16: N1-switch-1, adversary has access to oracles Oenc and Odec. Key space $\mathcal{K}_{mac}$ is the key space from M.mac. Lines 13 and 22 are different compared to lAE-N1^0, additionally lines 14, 15 and 25 from lAE-N1 are removed.

Using this game, we expand the probability:

$$\begin{aligned}\mathbf{Adv}^{\mathrm{lAE}}_{A,N} &= \Pr[\text{lAE-N1}^0_{A,N} = 0] - \Pr[\text{N1-switch-1}_{A,N} = 0] \\ &+ \Pr[\text{N1-switch-1}_{A,N} = 0] - \Pr[\text{lAE-N1}^1_{A,N} = 0].\end{aligned}$$

Next, we can rewrite $\Pr[\text{lAE-N1}_{A,N} = 0] - \Pr[\text{N1-switch-1}_{A,N} = 0]$ into a lMAC advantage. To do so, we define adversary B against lMAC in Figure 17. This adversary is playing game lMAC-PRF (Figure 10), and has access to A.

```
Adverary $B$
$L \leftarrow \emptyset$
for $j \in [1..N]$ :
    $k_j \xleftarrow{\$} \mathcal{K}_{enc}$
    $C_j \leftarrow \perp$
$b' \leftarrow$ run $A$
return $b'$

if $A$ calls Oracle Oenc$(j, l, m)$
if $C_j \neq \perp$ : return $\perp$
if $l \in L$ : return $\perp$
$L \leftarrow L \cup \{l\}$
$l_j \leftarrow l$
$c' \leftarrow$ E.enc$(k_j, l_j, m)$
$t \leftarrow$ Omac$(j, l_j, m)$
$c \leftarrow (c', t)$
$C_j \leftarrow c$
return $c$

if $A$ calls Oracle Odec$(j, c)$
if $C_j = \perp$ : return $\perp$
if $c = C_j$ : return $\perp$
$(c', t) \leftarrow c$
$m \leftarrow$ E.dec$(k_j, l_j, c')$
$passed \leftarrow$ Ovrf$(j, m, t')$
if $\neg passed$ : $m \leftarrow \perp$
return $m$
```

Figure 17: Adversary B, has access to A and oracles Omac and Ovrf. Key space $\mathcal{K}_{enc}$ is the key space from E.enc.

The runtime of B is that of A. For every Oenc query A makes, B computes E.enc once, and calls Omac once. For every Odec query A makes, B computes E.dec once and calls Ovrf once. Note that, alternatively, B could return 0 if *passed* is *true* to avoid having to do E.dec computations. To increase consistency with the other two cases, these computations are still made. We can see that $\Pr[\text{lMAC-PRF}^0_{B,N} = 0] = \Pr[\text{lAE-N1}^0_{A,N} = 0]$ as B perfectly simulates game lAE-N1 with $b = 0$ when its own b is 0. In addition, $\Pr[\text{lMAC-PRF}^1_{B,N} = 0] = \Pr[\text{N1-switch-1}_{A,N} = 0]$ as B perfectly simulates game N1-switch-1 whenever its own b is 1. As a result, we can rewrite our advantage to:

$$\begin{aligned}\mathbf{Adv}^{\mathrm{lAE}}_{A,N} &= \Pr[\text{lAE-N1}^0_{A,N} = 0] - \Pr[\text{N1-switch-1}_{A,N} = 0] \\ &+ \Pr[\text{N1-switch-1}_{A,N} = 0] - \Pr[\text{lAE-N1}^1_{A,N} = 0] \\ &= \Pr[\text{lMAC-PRF}^0_{B,N} = 0] - \Pr[\text{lMAC-PRF}^1_{B,N} = 0] \\ &+ \Pr[\text{N1-switch-1}_{A,N} = 0] - \Pr[\text{lAE-N1}^1_{A,N} = 0] \\ &= \mathbf{Adv}^{\mathrm{lMAC}}_{B,N} + \Pr[\text{N1-switch-1}_{A,N} = 0] - \Pr[\text{lAE-N1}^1_{A,N} = 0].\end{aligned}$$

To expand our advantage again, we define game N1-switch-2 in Figure 18. Apart from the Odec query, this game is equivalent to the first switch game. The Odec oracle from N1-switch-2 always returns $\perp$, it is written down more elaborately to include the event *bad*. This is event added to support a well-known proof tactic [6]. Our expanded advantage is:

$$\begin{aligned}\mathbf{Adv}^{\mathrm{lAE}}_{A,N} &= \mathbf{Adv}^{\mathrm{lMAC}}_{B,N} + \Pr[\text{N1-switch-1}_{A,N} = 0] - \Pr[\text{N1-switch-2}_{A,N} = 0] \\ &+ \Pr[\text{N1-switch-2}_{A,N} = 0] - \Pr[\text{lAE-N1}^1_{A,N} = 0].\end{aligned}$$

```
Game $\text{N1-switch-2}_{A,N}$
0 :  $L \leftarrow \emptyset$
1 :  $tag \xleftarrow{\$} Func(\mathcal{K}_{mac} \times \mathcal{L} \times \mathcal{M}, \mathcal{T})$
2 :  for $j \in [1..N]$ :
3 :      $k_j \xleftarrow{\$} \mathcal{K}$
4 :      $C_j \leftarrow \bot$
5 :  $b' \leftarrow A$
6 :  return $b'$

Oracle $\text{Oenc}(j, l, m)$
7 :  if $C_j \neq \bot$ : return $\bot$
8 :  if $l \in L$ : return $\bot$
9 :  $L \leftarrow L \cup \{l\}$
10 : $l_j \leftarrow l$
11 : $(k1, k2) \leftarrow k_j$
12 : $c' \leftarrow \text{E.enc}(k1, l_j, m)$
13 : $t \leftarrow tag(k2, l_j, m)$
14 : $c \leftarrow (c', t)$
15 : $C_j \leftarrow c$
16 : return $c$

Oracle $\text{Odec}(j, c)$
17 : if $C_j = \bot$ : return $\bot$
18 : if $c = C_j$ : return $\bot$
19 : $(k1, k2) \leftarrow k_j$
20 : $(c', t) \leftarrow c$
21 : $m \leftarrow \text{E.dec}(k1, l_j, c')$
22 : $t' \leftarrow tag(k2, l_j, m)$
23 : if $t \neq t'$ : $m \leftarrow \bot$
24 : else :
25 :     $bad \leftarrow true$
26 :     $m \leftarrow \bot$
27 : return $m$
```

Figure 18: N1-switch-2 game, adversary has access to oracles Oenc and Odec. Key space $\mathcal{K}_{mac}$ is the key space from M.mac. Line 24-26 are different compared to N1-switch-1.

As N1-switch-1 and N1-switch-2 are so called identical-until-*bad* [6], meaning they are equivalent as long as the event *bad* is not set to *true*, we know:

$$\Pr[\text{N1-switch-1}_{A,N} = 0] - \Pr[\text{N1-switch-2}_{A,N} = 0] \leq \Pr[bad = true].$$

As *bad* is set to *true* if, and only if, t=t', we can state $\Pr[bad = true] = \Pr[t = t']$. The adversary needs to provide tag t and ciphertext c', where ciphertext c' leads to a message m that is used as input to the *tag* function. The provided tag-ciphertext pair may not be the result of the encryption query corresponding to the provided user. Combined with the tidiness of the encryption, it is ensured that, for any sensible adversarial query, the message m cannot be the message which is encrypted for the provided user. This is because if m would be the encrypted message, the correct tag cannot be provided and thus no information can be gained with the query. Consequently, t and t' are only equal when the adversary is able to guess the output of *tag* for a message that is not encrypted for the provided user. The function *tag* is uniformly random so, with every fresh ciphertext, the probability that t and t' are equal is $\frac{1}{2^{\text{n}}}$. Summed over at most Q_d Odec queries we get $\Pr[t = t'] = \Pr[bad = true] \leq \frac{Q_d}{2^{\text{n}}}$ and thus, we can use $\Pr[\text{N2-switch-1}_{A,N} = 0] - \Pr[\text{N1-switch-2}_{A,N} = 0] \leq \Pr[bad = true] \leq \frac{Q_d}{2^{\text{n}}}$ to obtain:

$$\mathbf{Adv}^{\text{lAE}}_{A,N} \leq \mathbf{Adv}^{\text{lMAC}}_{B,N} + \Pr[\text{N1-switch-2}_{A,N} = 0] - \Pr[\text{lAE-N2}^{1}_{A,N} = 0] + \frac{Q_d}{2^{\text{n}}}.$$

We define game N1-switch-3 in Figure 19 to expand our advantage one last time. Switch game 3 is equivalent to switch game 2 but always returns lazily sampled random bits when the outcome of E.enc is valid. It might seem like there is a difference as $t$can no longer become $\bot$. This is not the case as, due to the chosen input spaces of tag, tag can only return $\bot$ whenever c' is already $\bot$. As a result, tag will never influence wether or not c on line 12 is $\bot$. We also simplify Odec as we no longer need the event *bad*. We use this game to expand our advantage to:

$$\begin{aligned}\mathbf{Adv}^{\text{lAE}}_{A,N} \leq \mathbf{Adv}^{\text{lMAC}}_{B,N} &+ \Pr[\text{N1-switch-2}_{A,N} = 0] - \Pr[\text{N1-switch-3}_{A,N} = 0] \\ &+ \Pr[\text{N1-switch-3}_{A,N} = 0] - \Pr[\text{lAE-N1}^{1}_{A,N} = 0] + \frac{Q_d}{2^{\text{n}}}.\end{aligned}$$

```
Game N1-switch-3$_{A,N}$
0 :  $L \leftarrow \emptyset$
1 :  for $j \in [1..N]$ :
2 :      $k_j \xleftarrow{\$} \mathcal{K}_{enc}$
3 :      $C_j \leftarrow \bot$
4 :  $b' \leftarrow A$
5 :  return $b'$

Oracle Oenc$(j, l, m)$
6 :  if $C_j \neq \bot$ : return $\bot$
7 :  if $l \in L$ : return $\bot$
8 :  $L \leftarrow L \cup \{l\}$
9 :  $l_j \leftarrow l$
10 : $c' \leftarrow$ E.enc$(kj, l_j, m)$
11 : $t \xleftarrow{\$} \{0,1\}^{\mathrm{n}}$
12 : $c \leftarrow (c', t)$
13 : if $c \neq \bot$ :
14 :     $c \xleftarrow{\$} \{0,1\}^{|c|}$
15 : $C_j \leftarrow c$
16 : return $c$

Oracle Odec$(j, c)$
18 : return $\bot$
```

Figure 19: N1-switch-3 game, adversary has access to oracles Oenc and Odec. Key space $\mathcal{K}_{enc}$ is the key space from E.enc. Line 14 and 15 are different compared to N1-switch-2, and Odec is simplified.

Pr[N1-switch-3$_{A,N}$ = 0] and Pr[lAE-N1$^1_{A,N}$ = 0] are equivalent by definition, giving:

$$\mathbf{Adv}^{\mathrm{lAE}}_{A,N} \leq \mathbf{Adv}^{\mathrm{lMAC}}_{B,N} + \Pr[\text{N1-switch-2}_{A,N} = 0] - \Pr[\text{N1-switch-3}_{A,N} = 0] + \frac{Q_d}{2^{\mathrm{n}}}.$$

Next, we can rewrite Pr[N1-switch-2$_{A,N}$ = 0] − Pr[N1-switch-3$_{A,N}$ = 0] into a lE advantage. To do so, we define adversary C against lE in Figure 20. This adversary is playing game lE-IND-\$-CPA (Figure 9), and has access to A.

```
Adverary $C$
0 :  $L \leftarrow \emptyset$
1 :  for $j \in [1..N]$ :
2 :      $C_j \leftarrow \bot$
3 :  $b' \leftarrow$ run $A$
4 :  return $b'$

if $A$ calls Oracle Oenc$(j, l, m)$
5 :  if $C_j \neq \bot$ : return $\bot$
6 :  if $l \in L$ : return $\bot$
7 :  $L \leftarrow L \cup \{l\}$
8 :  $l_j \leftarrow l$
9 :  $c' \leftarrow$ Oenc$(j, l_j, m)$
10 : $t \xleftarrow{\$} \{0,1\}^{\mathrm{n}}$
11 : $c \leftarrow (c', t)$
12 : $C_j \leftarrow c$
13 : return $c$

if $A$ calls Oracle Odec$(j, c)$
14 : return $\bot$
```

Figure 20: Adversary C, has access to A and oracle Oenc. Note the Oenc in line 9 refers to the encryption oracle Oenc that C has access to, not the oracle Oenc A has access to.

The runtime of C is that of A. For every Oenc query A makes, C makes one Oenc query. We can see that Pr[N1-switch-2$_{A,N}$ = 0] = Pr[lE-IND-\$-CPA$^0_{C,N}$ = 0] as C perfectly simulates N1-switch-2 when its own b is 0. When its own b is 1, C perfectly simulates N1-switch-3 giving

$\Pr[\text{N1-switch-3}_{A,N} = 0] = \Pr[\text{lE-IND-\$-CPA}^{1}_{C,N} = 0]$. This leads us to:

$$\begin{aligned}\mathbf{Adv}^{\text{lAE}}_{A,N} &\leq \mathbf{Adv}^{\text{lMAC}}_{B,N} + \Pr[\text{N1-switch-2}_{A,N} = 0] - \Pr[\text{N1-switch-3}_{A,N} = 0] + \frac{Q_d}{2^{\text{n}}}\\ &\leq \mathbf{Adv}^{\text{lMAC}}_{B,N} + \Pr[\text{lE-IND-\$-CPA}^{0}_{C,N} = 0] - \Pr[\text{lE-IND-\$-CPA}^{1}_{C,N} = 0] + \frac{Q_d}{2^{\text{n}}}\\ &\leq \mathbf{Adv}^{\text{lMAC}}_{B,N} + \mathbf{Adv}^{\text{lE}}_{C,N} + \frac{Q_d}{2^{\text{n}}}.\end{aligned}$$

Thus Proving:

$$\mathbf{Adv}^{\text{lAE}}_{A,N} \leq \mathbf{Adv}^{\text{lMAC}}_{B,N} + \mathbf{Adv}^{\text{lE}}_{C,N} + \frac{Q_d}{2^{\text{n}}}. \qquad \square$$

N2 and N3 Structurally, the three proofs are identical. For each case, the games, as well as the adversaries, are adjusted to AE.enc and AE.dec corresponding to the N-scheme. As a result, they differ in the three lines generating c and the three lines generating t'. As a example the games lAE-N1 in Figure 15, lAE-N2 in Figure 21 and lAE-N3 in Figure 27 only differ on lines 11 to 13 and lines 21 to 23. In addition the input spaces to the functions are different to facilitate this change. The argument leading to $\Pr[\mathit{bad} = \mathit{true}] \leq \frac{Q_d}{2^{\text{n}}}$ is different for the three cases as the tags are generated in a different way. To highlight this difference, the argument is put in between horizontal bars in the proofs for N2 and N3. The full proves are in Appendix A.

7 Use Cases

In this section, we will look at some use cases for the lAE. The most prominent use case is hybrid encryption as the idea of locks originated in this setting. In addition to this, the primitive might be useful whenever a new key is generated often as this will increase the number of ephemeral keys used and hence the chance on ephemeral key collisions. Below, we describe how the lAE can be used as a building block for hybrid encryption, as well as how it might be used in different settings.

7.1 Hybrid Encryption

To implement public-key encryption, a hybrid paradigm first formalized by Cramer and Shoup [7] is typically followed: To encrypt a message, an ephemeral key is generated using a randomized key encapsulation mechanism (KEM). This key is then used to encrypt the message using a deterministic data encapsulation mechanism (DEM). Both the KEM and DEM output their own ciphertexts, which are concatenated to form the public-key encryption ciphertext. Benefits of hybrid encryption is a separation of both primitives, as well as the possibility of variable-length messages, which can be lacking in other public-key encryption paradigms. GKP shows us how the lAE can be used in hybrid encryption. The KEM, as usual, generates an ephemeral key and encapsulates this key to create the first part of the ciphertext. Afterwards, the lAE uses the encapsulation as a lock and takes the ephemeral key and the message to instantiate the DEM. GKP also prove this construction to be secure whenever locks do not repeat. Our lAE does not need to be altered to be used in this setting as the ephemeral key is used once.

7.2 Other Use Cases

Whenever an authenticated encryption primitive uses a key only once, using lAE may decrease the degradation of security bounds in a multi-user setting. Even when the primitive uses the same key multiple times, using lAE may still decrease the degradation of security bounds whenever the key is changed often. One example of such a use case is the Messaging Layer Security protocol [8]. Generating a new key is necessary in this protocol whenever a member enters or leaves the group to ensure only current group members can read messages. In addition to this, new keys may be generated more often, depending on the implementation of the protocol. When the key is used multiple times, the lAE primitive most likely needs to be slightly altered. Section 9 describes this alteration further.

8 Related Work

As mentioned before, the generic composition of authenticated encryption was first studied by Bellare and Namprempre [3]. The three most common composition modes encrypt-and-MAC, encrypt-then-MAC and MAC-then-encrypt were introduced and evaluated using a probabilistic encryption block. They find generic construction to be secure when using the encrypt-then-MAC method. NRS further investigated these modes of composition and state that the type of encryption primitive used, as well as the required end-result, influences which compositions are secure. They investigated ways to compose a nonce-based authenticated encryption scheme. Using IV-based encryption, 8 schemes are proven secure. Using nonce-based encryption 3 schemes, related to encrypt-and-MAC, encrypt-then-MAC and MAC-then-encrypt, are proven secure.

the study of symmetric cryptographic primitives is usually done in a single-user setting, where one user that uses one key is considered. In reality, most systems have multiple users that are generate their own keys. To account for this, the notion of multi-user security, where one considers multiple users all generating their own keys, was first introduced by Bellare, Boldyreva and Micali [9]. Security bounds found in a single-user setting often degrade in a multi-user setting as noted by Biham [10]. He finds that, in a multi-user setting, the strength of a cipher can not exceed the square root of the key size. To compensate for this degradation, one can expand the size of the key. This is not always feasible so GKP introduce locks as an alternative to key expansion. They show how ephemeral key collisions in a multi-user setting can lead to insecurities, as well as how one can augment security primitives with locks. Afterwards, a composition of an augmented MAC and an augmented DEM is shown be suitable for hybrid encryption.

Next we will discuss some constructions relating to lAE, open problems relating to these constructions can be found in Section 9. A similar construction to locks, called “id”, is used to construct a sponge-based PRF [11] and to construct a PRF out of a permutation [12]. Just like the lock, this id is bound to the user to prevent degradation of security bounds due to user key collisions. The first construction assumes the id to be unique while the second allows the id to be shared between multiple users. The security bound of both constructions are proven to have little degradation in a multi-user setting when the underlying primitives are ideal.

As an alternative to generic composition, authenticated encryption can also be composed non-generically. In this fashion, Rogaway, Bellare and Black propose OCB as a non-generic authenticated encryption construction [13]. OCB is highly parallelizable and is cheaper in computation compared to generic construction. In the basic form, it cannot take in associated data but an extension has been given to allow for this. It is proven secure whenever the underlying block cipher is secure. McGrew and Viega propose Galois/Counter Mode, GCM, as a non-generic authenticated encryption construction [14]. It is a highly efficient mode of operation, also due to its parallelizability. furthermore, GCM incorporates counter mode using a even length block cipher and supports the usage of associated data. In a addition to authenticated encryption, it can also be used as a standalone MAC function. Other non-generic modes of operation exist as well, but these two are the most popular.

9 Conclusion

In this thesis, we studied the security of generic composition of authenticated encryption using locks. In order to do this, we first defined a new security primitive, named lock-based authenticated encryption. Three different ways in which we can compose this new primitive using an lock-based encryption primitive and a lock-based mac function were considered, namely encrypt-and-MAC, encrypt-then-MAC and MAC-then-encrypt. All three of these compositions were proven secure, whenever the underlying primitives are secure. Additionally, we explored some use cases of our new primitive. Most predominately, we explained how lock-based authenticated encryption can be used to instantiate a DEM in hybrid encryption. The results of this research has some open problems we will discuss below. Wherever possible we will also give directions on what needs to be done in order to further investigate these problems.

Other N schemes NRS finds 20 possible N-schemes, of which 3 are proven secure. In this thesis, the only N-schemes evaluated where alterations to the ones proven secure by NRS. To more thoroughly investigate the security of generic lAE compositions, the other N-schemes should also be proven secure or be proven insecure with a counterexample. The amount of possible generic compositions gets bigger if the authenticated encryption scheme has more inputs. Without incorporating AD, only 10 out of 20 possible N-schemes remain. Three of these schemes were proven secure in this work, leaving 7 schemes to be evaluated.

Adding associated Data In this work, the lAE construction was only evaluated without associated data (AD). To incorporate AD, a slight modification should be made to the definition of the lAE. With AD added, there will be 20 possible N-schemes, three of which will relate to the secure schemes from NRS. These three schemes should be prioritized as they have the highest likelihood of being secure. For a more rigorous analysis, all 20 schemes should be evaluated.

Evaluating lAE with multiple uses The evaluation of the lAE construction was limited to a key that is used once. A more in-depth analysis could evaluate the construction in a setting where a key can be used multiple (But still limited) times. In order to maintain security, one would likely need to alter the lAE definition to also incorporate nonces. As the scheme will then have an additional input, a new set of possible compositions should be generated and evaluated.

Augmenting non-generic AE constructions with locks We found adaptation using locks to be secure for generic composition of authenticated encryption. As a next step, one could look at wether non-generic AE constructions can be augmented with locks. When looking at the examples mentioned in Section 8, the augmented versions will most likely have lower computational cost when compared to generic composition, as well as be highly parallelizable. When proven secure, this will lead to more efficient instantiations of the lAE.

Instantiating the lE and lMAC We discussed how an lEA can be constructed from an lE and a lMAC. Further research could look how these two building blocks can be implemented. The two id-based PRF functions discussed in Section 8 might be used as an lE. In this case, the lock value might be used as the id as the concepts are very similar. Furthermore, a nonce-based encryption primitive or nonce-based MAC function might be leveraged as an lE or a lMAC respectively. In this case, the lock value might be used as a nonce because we assume globally unique locks that are used only once.

References

[1] F. Giacon, E. Kiltz, and B. Poettering, "Hybrid encryption in a multi-user setting, revisited," 2018, pp. 159–189. DOI: 10.1007/978-3-319-76578-5_6.

[2] C. Namprempre, P. Rogaway, and T. Shrimpton, "Reconsidering generic composition," 2014, pp. 257–274. DOI: 10.1007/978-3-642-55220-5_15.

[3] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," 2000, pp. 531–545. DOI: 10.1007/3-540-44448-3_41.

[4] C. Cremers, A. Dax, C. Jacomme, and M. Zhao, "Automated analysis of protocols that use authenticated encryption: How subtle AEAD differences can impact protocol security," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 5935–5952, ISBN: 978-1-939133-37-3.

[5] F. Berti, O. Pereira, and T. Peters, "Reconsidering generic composition: The tag-then-encrypt case," 2018, pp. 70–90. DOI: 10.1007/978-3-030-05378-9_4.

[6] M. Bellare and P. Rogaway, "The security of triple encryption and a framework for code-based game-playing proofs," 2006, pp. 409–426. DOI: 10.1007/11761679_25.

[7] R. Cramer and V. Shoup, "Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack," vol. 33, no. 1, pp. 167–226, 2003.

[8] R. Barnes, B. Beurdouche, R. Robert, J. Millican, E. Omara, and K. Cohn-Gordon, *The Messaging Layer Security (MLS) Protocol*, RFC 9420, 2023. DOI: 10.17487/RFC9420. [Online]. Available: https://www.rfc-editor.org/info/rfc9420.

[9] M. Bellare, A. Boldyreva, and S. Micali, "Public-key encryption in a multi-user setting: Security proofs and improvements," 2000, pp. 259–274. DOI: 10.1007/3-540-45539-6_18.

[10] E. Biham, "How to decrypt or even substitute des-encrypted messages in 228 steps," *Information Processing Letters*, vol. 84, no. 3, pp. 117–124, 2002, ISSN: 0020-0190. DOI: https://doi.org/10.1016/S0020-0190(02)00269-7.

[11] A. Bhattacharjee, R. Bhaumik, and M. Nandi, *A sponge-based PRF with good multi-user security*, Cryptology ePrint Archive, Report 2022/1146, https://eprint.iacr.org/2022/1146, 2022.

[12] C. Lefevre, Y. Belkheyar, and J. Daemen, *Kirby: A robust permutation-based prf construction*, Cryptology ePrint Archive, Report 2023/1520, //https://eprint.iacr.org/2023/1520, 2023.

[13] P. Rogaway, M. Bellare, J. Black, and T. Krovetz, "OCB: A block-cipher mode of operation for efficient authenticated encryption," 2001, pp. 196–205. DOI: 10.1145/501983.502011.

[14] D. A. McGrew and J. Viega, "The security and performance of the galois/counter mode (gcm) of operation," in *International Conference on Cryptology in India*, Springer, 2004, pp. 343–355.

A Proof of N2 and N3

In this appendix, the full proof of case N2 and N3 can be found.

N2 First, we repeat theorem 1 specifically for N2:

Theorem 3. *Let lAE be constructed from lMAC and lE as described in Figure 13. Let ciphertext space $\mathcal{C}$ from the lE be a subset of message space $\mathcal{M}$ from the lMAC and let lMAC and lE have a shared lock space. Then, for any number of users N and any lAE adversary A that poses at most Q_e many Oenc queries, and at most Q_d many Odec queries, there exist a lMAC adversary B and a lE adversary C such that:*

$$\mathbf{Adv}^{\mathrm{lAE}}_{A,N} \leq \mathbf{Adv}^{\mathrm{lMAC}}_{B,N} + \mathbf{Adv}^{\mathrm{lE}}_{C,N} + \frac{Q_d}{2^{\mathrm{n}}},$$

where n *is the output length of the lMAC in bits. The running time of B is at most that of A plus the time required to run Q_e many* E.enc *encapsulations and Q_d many* E.dec *decapsulations. The running time of C is at most that of A. Additionally, B makes at most Q_e many Omac queries and at most Q_d many Ovrf queries and C makes at most Q_e many Oenc queries.*

Within this theorem, both Q_e and Q_d refer to the total queries the adversary is allowed to make, not the queries per user. As a result, Q_e is limited by N.

Proof. To prove this theorem, we start by defining game lAE-N2 in Figure 21. This game is the game lAE-IND-\$-CCA (Figure 8), with AE.enc and AE.dec substituted with the N2 algorithms from Figure 13.

```
Game lAE-N2$^b_{A,N}$
$L \leftarrow \emptyset$
for $j \in [1..N]$ :
    $k_j \xleftarrow{\$} \mathcal{K}$
    $C_j \leftarrow \perp$
$b' \leftarrow A$
return $b'$

Oracle Oenc$(j, l, m)$
if $C_j \neq \perp$ : return $\perp$
if $l \in L$ : return $\perp$
$L \leftarrow L \cup \{l\}$
$l_j \leftarrow l$
$(k1, k2) \leftarrow k_j$
$c' \leftarrow$ E.enc$(k1, l_j, m)$
$t \leftarrow$ M.mac$(k2, l_j, c')$
$c \leftarrow (c', t)$
if $b = 1 \wedge c \neq \perp$ :
    $c \xleftarrow{\$} \{0,1\}^{|c|}$
$C_j \leftarrow c$
return $c$

Oracle Odec$(j, c)$
if $C_j = \perp$ : return $\perp$
if $c = C_j$ : return $\perp$
$(k1, k2) \leftarrow k_j$
$(c', t) \leftarrow c$
$m \leftarrow$ E.dec$(k1, l_j, c')$
$t' \leftarrow$ M.mac$(k2, l_j, c')$
if $t \neq t'$ : $m \leftarrow \perp$
if $b = 1$ : $m \leftarrow \perp$
return $m$
```

Figure 21: lAE-N2 game, adversary has access to oracles Oenc and Odec.

By definition, this gives us

$$\mathbf{Adv}^{\mathrm{lAE}}_{A,N} = \Pr[\mathrm{lAE\text{-}N2}^0_{A,N} = 0] - \Pr[\mathrm{lAE\text{-}N2}^1_{A,N} = 0].$$

Next we define game N2-switch-1 in Figure 22. The only difference between this game and game lAE-N2^0 is the fact that N2-switch-1 uses the uniformly random function *tag*, instead of the lMAC. To define this function we write *Func*$(\mathcal{K} \times \mathcal{L} \times \mathcal{C}, \mathcal{T})$ to denote the set of all functions from the key space of the MAC $\mathcal{K}$, the shared lock space $\mathcal{L}$ and ciphertext space from E.enc $\mathcal{C}$ to the tag space $\mathcal{T}$. We define this function specifically as we want the tags resulting from computations in oracle Oenc to match with those in oracle Odec. When the input of *tag* is outside its domain, it will return ⊥.

```
Game N2-switch-1_{A,N}
0 :  $L \leftarrow \emptyset$
1 :  $tag \xleftarrow{\$} Func(\mathcal{K}_{mac} \times \mathcal{L} \times \mathcal{C}, \mathcal{T})$
2 :  for $j \in [1..N]$ :
3 :      $k_j \xleftarrow{\$} \mathcal{K}$
4 :      $C_j \leftarrow \perp$
5 :  $b' \leftarrow A$
6 :  return $b'$

Oracle Oenc(j, l, m)
7 :  if $C_j \neq \perp$ : return $\perp$
8 :  if $l \in L$ : return $\perp$
9 :  $L \leftarrow L \cup \{l\}$
10 : $l_j \leftarrow l$
11 : $(k1, k2) \leftarrow k_j$
12 : $c' \leftarrow \text{E.enc}(k1, l_j, m)$
13 : $t \leftarrow tag(k2, l_j, c')$
14 : $c \leftarrow (c', t)$
15 : $C_j \leftarrow c$
16 : return $c$

Oracle Odec(j, c)
17 : if $C_j = \perp$ : return $\perp$
18 : if $c = C_j$ : return $\perp$
19 : $(k1, k2) \leftarrow k_j$
20 : $(c', t) \leftarrow c$
21 : $m \leftarrow \text{E.dec}(k1, l_j, c')$
22 : $t' \leftarrow tag(k2, l_j, c')$
23 : if $t \neq t'$ : $m \leftarrow \perp$
24 : return $m$
```

Figure 22: N2-switch-1, adversary has access to oracles Oenc and Odec. Key space $\mathcal{K}_{mac}$ is the key space from M.mac. Lines 13 and 22 are different compared to lAE-N2^0, additionally lines 14, 15 and 25 from lAE-N2 are removed.

Using this game, we expand the probability:

$$\begin{aligned}\mathbf{Adv}^{\text{lAE}}_{A,N} &= \Pr[\text{lAE-N2}^0_{A,N} = 0] - \Pr[\text{N2-switch-1}_{A,N} = 0] \\ &+ \Pr[\text{N2-switch-1}_{A,N} = 0] - \Pr[\text{lAE-N2}^1_{A,N} = 0].\end{aligned}$$

Next, we can rewrite $\Pr[\text{lAE-N2}_{A,N} = 0] - \Pr[\text{N2-switch-1}_{A,N} = 0]$ into a lMAC advantage. To do so, we define adversary B against lMAC in Figure 23. This adversary is playing game lMAC-PRF (Figure 10), and has access to A.

```
Adverary B
$L \leftarrow \emptyset$
for $j \in [1..N]$ :
    $k_j \xleftarrow{\$} \mathcal{K}_{enc}$
    $C_j \leftarrow \perp$
$b' \leftarrow$ run $A$
return $b'$

if A calls Oracle Oenc(j, l, m)
if $C_j \neq \perp$ : return $\perp$
if $l \in L$ : return $\perp$
$L \leftarrow L \cup \{l\}$
$l_j \leftarrow l$
$c' \leftarrow \text{E.enc}(k_j, l_j, m)$
$t \leftarrow \text{Omac}(j, l_j, c')$
$c \leftarrow (c', t)$
$C_j \leftarrow c$
return $c$

if A calls Oracle Odec(j, c)
if $C_j = \perp$ : return $\perp$
if $c = C_j$ : return $\perp$
$(c', t) \leftarrow c$
$m \leftarrow \text{E.dec}(k_j, l_j, c')$
$passed \leftarrow \text{Ovrf}(j, c', t')$
if $\neg passed$ : $m \leftarrow \perp$
return $m$
```

Figure 23: Adversary B, has access to A and oracles Omac and Ovrf. Key space $\mathcal{K}_{enc}$ is the key space from E.enc.

The runtime of B is that of A. For every Oenc query A makes, B computes E.enc once, and calls Omac once. For every Odec query A makes, B computes E.dec once and calls Ovrf once. Note that, alternatively, B could return 0 if *passed* is *true* to avoid having to do E.dec computations. To increase consistency with the other two cases, these computations are still made. We can see that $\Pr[\text{lMAC-PRF}^0_{B,N} = 0] = \Pr[\text{lAE-N2}^0_{A,N} = 0]$ as B perfectly simulates game lAE-N2 with $b = 0$ when its own b is 0. In addition, $\Pr[\text{lMAC-PRF}^1_{B,N} = 0] = \Pr[\text{N2-switch-1}_{A,N} = 0]$ as B perfectly simulates game N2-switch-1 whenever its own b is 1. As a result, we can rewrite our advantage to:

$$\begin{aligned}\mathbf{Adv}^{\text{lAE}}_{A,N} &= \Pr[\text{lAE-N2}^0_{A,N} = 0] - \Pr[\text{N2-switch-1}_{A,N} = 0] \\ &+ \Pr[\text{N2-switch-1}_{A,N} = 0] - \Pr[\text{lAE-N2}^1_{A,N} = 0] \\ &= \Pr[\text{lMAC-PRF}^0_{B,N} = 0] - \Pr[\text{lMAC-PRF}^1_{B,N} = 0] \\ &+ \Pr[\text{N2-switch-1}_{A,N} = 0] - \Pr[\text{lAE-N2}^1_{A,N} = 0] \\ &= \mathbf{Adv}^{\text{lMAC}}_{B,N} + \Pr[\text{N2-switch-1}_{A,N} = 0] - \Pr[\text{lAE-N2}^1_{A,N} = 0].\end{aligned}$$

To expand our advantage again, we define game N2-switch-2 in Figure 24. Apart from the Odec query, this game is equivalent to the first switch game. The Odec oracle from N2-switch-2 always returns $\perp$, it is written down more elaborately to include the event *bad*. This is event added to support a well-known proof tactic [6]. Our expanded advantage is:

$$\begin{aligned}\mathbf{Adv}^{\text{lAE}}_{A,N} &= \mathbf{Adv}^{\text{lMAC}}_{B,N} + \Pr[\text{N2-switch-1}_{A,N} = 0] - \Pr[\text{N2-switch-2}_{A,N} = 0] \\ &+ \Pr[\text{N2-switch-2}_{A,N} = 0] - \Pr[\text{lAE-N2}^1_{A,N} = 0].\end{aligned}$$

```
Game N2-switch-2$_{A,N}$
0 :  $L \leftarrow \emptyset$
1 :  $tag \xleftarrow{\$} Func(\mathcal{K}_{mac} \times \mathcal{L} \times \mathcal{C}, \mathcal{T})$
2 :  for $j \in [1..N]$ :
3 :      $k_j \xleftarrow{\$} \mathcal{K}$
4 :      $C_j \leftarrow \perp$
5 :  $b' \leftarrow A$
6 :  return $b'$
```

```
Oracle Oenc$(j, l, m)$
7 :  if $C_j \neq \perp$ : return $\perp$
8 :  if $l \in L$ : return $\perp$
9 :  $L \leftarrow L \cup \{l\}$
10 : $l_j \leftarrow l$
11 : $(k1, k2) \leftarrow k_j$
12 : $c' \leftarrow$ E.enc$(k1, l_j, m)$
13 : $t \leftarrow tag(k2, l_j, c')$
14 : $c \leftarrow (c', t)$
15 : $C_j \leftarrow c$
16 : return $c$
```

```
Oracle Odec$(j, c)$
17 : if $C_j = \perp$ : return $\perp$
18 : if $c = C_j$ : return $\perp$
19 : $(k1, k2) \leftarrow k_j$
20 : $(c', t) \leftarrow c$
21 : $m \leftarrow$ E.dec$(k1, l_j, c')$
22 : $t' \leftarrow tag(k2, l_j, c')$
23 : if $t \neq t'$ : $m \leftarrow \perp$
24 : else :
25 :     $bad \leftarrow true$
26 :     $m \leftarrow \perp$
27 : return $m$
```

Figure 24: N2-switch-2 game, adversary has access to oracles Oenc and Odec. Key space $\mathcal{K}_{mac}$ is the key space from M.mac. Line 24-26 are different compared to N2-switch-1.

As N2-switch-1 and N2-switch-2 are so called identical-until-*bad* [6], meaning they are equivalent as long as the event *bad* is not set to *true*, we know:

$$\Pr[\text{N2-switch-1}_{A,N} = 0] - \Pr[\text{N2-switch-2}_{A,N} = 0] \leq \Pr[bad = true].$$

Below thick lines indicate the technical difference of current proof with the one of N1 in Section 6

As *bad* is set to *true* if, and only if, $t{=}t'$, we can state $\Pr[bad = true] = \Pr[t = t']$. The adversary needs to provide tag t and ciphertext c' for the *tag* function, where the provided tag-ciphertext pair may not be the result of the encryption query corresponding to the provided user. Consequently, t and t' are only equal when the adversary is able to guess the output of *tag* for a ciphertext that is not encrypted for the provided user. The function *tag* is uniformly random so, with every fresh ciphertext, the probability that t and t' are equal is $\frac{1}{2^{\mathrm{n}}}$. Summed over at most Q_d Odec queries we get $\Pr[t = t'] = \Pr[bad = true] \leq \frac{Q_d}{2^{\mathrm{n}}}$ and thus, we can use $\Pr[\text{N2-switch-1}_{A,N} = 0] - \Pr[\text{N2-switch-2}_{A,N} = 0] \leq \Pr[bad = true] \leq \frac{Q_d}{2^{\mathrm{n}}}$ to obtain:

$$\mathbf{Adv}^{\text{lAE}}_{A,N} \leq \mathbf{Adv}^{\text{lMAC}}_{B,N} + \Pr[\text{N2-switch-2}_{A,N} = 0] - \Pr[\text{lAE-N2}^{1}_{A,N} = 0] + \frac{Q_d}{2^{\mathrm{n}}}.$$

We define game N2-switch-3 in Figure 25 to expand our advantage one last time. Switch game 3 is equivalent to switch game 2 but always returns lazily sampled random bits when the outcome of E.enc is valid. It might seem like there is a difference as $t$can no longer become $\perp$. This is not the case as, due to the chosen input spaces of tag, tag can only return $\perp$ whenever c' is already $\perp$. As a result, tag will never influence wether or not c on line 12 is $\perp$. We also simplify Odec as we no longer need the event *bad*. We use this game to expand our advantage to:

$$\begin{aligned}\mathbf{Adv}^{\text{lAE}}_{A,N} \leq \mathbf{Adv}^{\text{lMAC}}_{B,N} &+ \Pr[\text{N2-switch-2}_{A,N} = 0] - \Pr[\text{N2-switch-3}_{A,N} = 0] \\ &+ \Pr[\text{N2-switch-3}_{A,N} = 0] - \Pr[\text{lAE-N2}^{1}_{A,N} = 0] + \frac{Q_d}{2^{\mathrm{n}}}.\end{aligned}$$

```
Game N2-switch-3$_{A,N}$
0 :  $L \leftarrow \emptyset$
1 :  for $j \in [1..N]$ :
2 :      $k_j \xleftarrow{\$} \mathcal{K}_{enc}$
3 :      $C_j \leftarrow \perp$
4 :  $b' \leftarrow A$
5 :  return $b'$
```

```
Oracle Oenc$(j, l, m)$
6 :   if $C_j \neq \perp$ : return $\perp$
7 :   if $l \in L$ : return $\perp$
8 :   $L \leftarrow L \cup \{l\}$
9 :   $l_j \leftarrow l$
10 :  $c' \leftarrow$ E.enc$(kj, l_j, m)$
11 :  $t \xleftarrow{\$} \{0,1\}^{\mathrm{n}}$
12 :  $c \leftarrow (c', t)$
13 :  if $c \neq \perp$ :
14 :      $c \xleftarrow{\$} \{0,1\}^{|c|}$
15 :  $C_j \leftarrow c$
16 :  return $c$
```

```
Oracle Odec$(j, c)$
18 :  return $\perp$
```

Figure 25: N2-switch-3 game, adversary has access to oracles Oenc and Odec. Key space $\mathcal{K}_{enc}$ is the key space from E.enc. Line 14 and 15 are different compared to N2-switch-2, and Odec is simplified.

$\Pr[\text{N2-switch-3}_{A,N} = 0]$ and $\Pr[\text{lAE-N2}^{1}_{A,N} = 0]$ are equivalent by definition, giving:

$$\mathbf{Adv}^{\text{lAE}}_{A,N} \leq \mathbf{Adv}^{\text{lMAC}}_{B,N} + \Pr[\text{N2-switch-2}_{A,N} = 0] - \Pr[\text{N2-switch-3}_{A,N} = 0] + \frac{Q_d}{2^{\mathrm{n}}}.$$

Next, we can rewrite $\Pr[\text{N2-switch-2}_{A,N} = 0] - \Pr[\text{N2-switch-3}_{A,N} = 0]$ into a lE advantage. To do so, we define adversary C against lE in Figure 26. This adversary is playing game lE-IND-\$-CPA (Figure 9), and has access to A.

```
Adverary $C$
0 :  $L \leftarrow \emptyset$
1 :  for $j \in [1..N]$ :
2 :      $C_j \leftarrow \perp$
3 :  $b' \leftarrow$ run $A$
4 :  return $b'$

if $A$ calls Oracle Oenc$(j, l, m)$
5 :  if $C_j \neq \perp$ : return $\perp$
6 :  if $l \in L$ : return $\perp$
7 :  $L \leftarrow L \cup \{l\}$
8 :  $l_j \leftarrow l$
9 :  $c' \leftarrow \text{Oenc}(j, l_j, m)$
10 : $t \xleftarrow{\$} \{0,1\}^{\mathrm{n}}$
11 : $c \leftarrow (c', t)$
12 : $C_j \leftarrow c$
13 : return $c$

if $A$ calls Oracle Odec$(j, c)$
14 : return $\perp$
```

Figure 26: Adversary C, has access to A and oracle Oenc. Note the Oenc in line 9 refers to the encryption oracle Oenc that C has access to, not the oracle Oenc A has access to.

The runtime of C is that of A. For every Oenc query A makes, C makes one Oenc query. We can see that $\Pr[\text{N2-switch-2}_{A,N} = 0] = \Pr[\text{lE-IND-\$-CPA}^0_{C,N} = 0]$ as C perfectly simulates N2-switch-2 when its own b is 0. When its own b is 1, C perfectly simulates N2-switch-3 giving $\Pr[\text{N2-switch-3}_{A,N} = 0] = \Pr[\text{lE-IND-\$-CPA}^1_{C,N} = 0]$. This leads to:

$$\begin{aligned}\mathbf{Adv}^{\text{lAE}}_{A,N} &\leq \mathbf{Adv}^{\text{lMAC}}_{B,N} + \Pr[\text{N2-switch-2}_{A,N} = 0] - \Pr[\text{N2-switch-3}_{A,N} = 0] + \frac{Q_d}{2^{\mathrm{n}}} \\ &\leq \mathbf{Adv}^{\text{lMAC}}_{B,N} + \Pr[\text{lE-IND-\$-CPA}^0_{C,N} = 0] - \Pr[\text{lE-IND-\$-CPA}^1_{C,N} = 0] + \frac{Q_d}{2^{\mathrm{n}}} \\ &\leq \mathbf{Adv}^{\text{lMAC}}_{B,N} + \mathbf{Adv}^{\text{lE}}_{C,N} + \frac{Q_d}{2^{\mathrm{n}}}.\end{aligned}$$

Thus Proving:

$$\mathbf{Adv}^{\text{lAE}}_{A,N} \leq \mathbf{Adv}^{\text{lMAC}}_{B,N} + \mathbf{Adv}^{\text{lE}}_{C,N} + \frac{Q_d}{2^{\mathrm{n}}}. \qquad \square$$

N3 First, we repeat theorem 1 specifically for N3:

Theorem 4. *Let lAE be constructed from lMAC and lE as described in Figure 14. Let ciphertext space $\mathcal{C}$ from the lE be a subset of message space $\mathcal{M}$ from the lMAC and let lMAC and lE have a shared lock space. Then, for any number of users N and any lAE adversary A that poses at most Q_e many Oenc queries, and at most Q_d many Odec queries, there exist a lMAC adversary B and a lE adversary C such that:*

$$\mathbf{Adv}^{\text{lAE}}_{A,N} \leq \mathbf{Adv}^{\text{lMAC}}_{B,N} + \mathbf{Adv}^{\text{lE}}_{C,N} + \frac{Q_d}{2^{\mathrm{n}}},$$

where n *is the output length of the lMAC in bits. The running time of B is at most that of A plus the time required to run Q_e many* E.enc *encapsulations and Q_d many* E.dec *decapsulations. The running time of C is at most that of A. Additionally, B makes at most Q_e many Omac queries and at most Q_d many Ovrf queries and C makes at most Q_e many Oenc queries.*

Within this theorem, both Q_e and Q_d refer to the total queries the adversary is allowed to make, not the queries per user. As a result, Q_e is limited by N.

Proof. To prove this theorem, we start by defining game lAE-N3 in Figure 27. This game is the game lAE-IND-\$-CCA (Figure 8), with AE.enc and AE.dec substituted with the N3 algorithms from Figure 14.

```
Game $\text{lAE-N3}^b_{A,N}$
$L \leftarrow \emptyset$
for $j \in [1..N]$ :
    $k_j \xleftarrow{\$} \mathcal{K}$
    $C_j \leftarrow \bot$
$b' \leftarrow A$
return $b'$
```

```
Oracle $\text{Oenc}(j, l, m)$
if $C_j \neq \bot$ : return $\bot$
if $l \in L$ : return $\bot$
$L \leftarrow L \cup \{l\}$
$l_j \leftarrow l$
$(k1, k2) \leftarrow k_j$
$t \leftarrow \text{M.mac}(k2, l_j, m)$
$m' \leftarrow m \| t$
$c \leftarrow \text{E.enc}(k1, l_j, m')$
if $b = 1 \wedge c \neq \bot$ :
    $c \xleftarrow{\$} \{0,1\}^{|c|}$
$C_j \leftarrow c$
return $c$
```

```
Oracle $\text{Odec}(j, c)$
if $C_j = \bot$ : return $\bot$
if $c = C_j$ : return $\bot$
$(k1, k2) \leftarrow k_j$
$m' \leftarrow \text{E.dec}(k1, l_j, c)$
$(m, t) \leftarrow m'$
$t' \leftarrow \text{M.mac}(k2, l_j, m)$
if $t \neq t'$ : $m \leftarrow \bot$
if $b = 1$ : $m \leftarrow \bot$
return $m$
```

Figure 27: lAE-N3 game, adversary has access to oracles Oenc and Odec.

By definition, this gives us

$$\mathbf{Adv}^{\text{lAE}}_{A,N} = \Pr[\text{lAE-N3}^0_{A,N} = 0] - \Pr[\text{lAE-N3}^1_{A,N} = 0].$$

Next we define game N3-switch-1 in Figure 28. The only difference between this game and game lAE-N3^0 is the fact that N3-switch-1 uses the uniformly random function *tag*, instead of the lMAC. To define this function we write $\mathit{Func}(\mathcal{K} \times \mathcal{L} \times \mathcal{M}, \mathcal{T})$ to denote the set of all functions from the key space of the MAC $\mathcal{K}$, the shared lock space $\mathcal{L}$ and the message space $\mathcal{M}$ to the tag space $\mathcal{T}$. We define this function specifically as we want the tags resulting from computations in oracle Oenc to match with those in oracle Odec. When the input of *tag* is outside its domain, it will return $\bot$.

```
**Game** N3-switch-$1_{A,N}$
0 :  $L \leftarrow \emptyset$
1 :  $tag \xleftarrow{\$} Func(\mathcal{K}_{mac} \times \mathcal{L} \times \mathcal{M}, \mathcal{T})$
2 :  for $j \in [1..N]$ :
3 :      $k_j \xleftarrow{\$} \mathcal{K}$
4 :      $C_j \leftarrow \bot$
5 :  $b' \leftarrow A$
6 :  return $b'$
```

```
**Oracle** Oenc$(j, l, m)$
7 :  if $C_j \neq \bot$ : return $\bot$
8 :  if $l \in L$ : return $\bot$
9 :  $L \leftarrow L \cup \{l\}$
10 : $l_j \leftarrow l$
11 : $(k1, k2) \leftarrow k_j$
12 : $t \leftarrow tag(k2, l_j, m)$
13 : $m' \leftarrow m \| t$
14 : $c \leftarrow$ E.enc$(k1, l_j, m')$
15 : $C_j \leftarrow c$
16 : return $c$
```

```
**Oracle** Odec$(j, c)$
17 : if $C_j = \bot$ : return $\bot$
18 : if $c = C_j$ : return $\bot$
19 : $(k1, k2) \leftarrow k_j$
20 : $m' \leftarrow$ E.dec$(k1, l_j, c)$
21 : $(m, t) \leftarrow m'$
22 : $t' \leftarrow tag(k2, l_j, m)$
23 : if $t \neq t'$ : $m \leftarrow \bot$
24 : return $m$
```

Figure 28: N3-switch-1, adversary has access to oracles Oenc and Odec. Key space $\mathcal{K}_{mac}$ is the key space from M.mac. Lines 12 and 22 are different compared to lAE-N3^0, additionally lines 14, 15 and 25 from lAE-N3 are removed.

Using this game, we expand the probability:

$$\begin{aligned}\mathbf{Adv}^{\text{lAE}}_{A,N} &= \Pr[\text{lAE-N}3^0_{A,N} = 0] - \Pr[\text{N3-switch-}1_{A,N} = 0] \\ &+ \Pr[\text{N3-switch-}1_{A,N} = 0] - \Pr[\text{lAE-N}3^1_{A,N} = 0].\end{aligned}$$

Next, we can rewrite $\Pr[\text{lAE-N}3_{A,N} = 0] - \Pr[\text{N3-switch-}1_{A,N} = 0]$ into a lMAC advantage. To do so, we define adversary B against lMAC in Figure 29. This adversary is playing game lMAC-PRF (Figure 10), and has access to A.

```
**Adverary** $B$
$L \leftarrow \emptyset$
for $j \in [1..N]$ :
    $k_j \xleftarrow{\$} \mathcal{K}_{enc}$
    $C_j \leftarrow \bot$
$b' \leftarrow$ run $A$
return $b'$
```

```
if $A$ calls **Oracle** Oenc$(j, l, m)$
if $C_j \neq \bot$ : return $\bot$
if $l \in L$ : return $\bot$
$L \leftarrow L \cup \{l\}$
$l_j \leftarrow l$
$t \leftarrow$ Omac$(k_j, l_j, m)$
$m' \leftarrow m \| t$
$c \leftarrow$ E.enc$(k_j, l_j, m')$
$C_j \leftarrow c$
return $c$
```

```
if $A$ calls **Oracle** Odec$(j, c)$
if $C_j = \bot$ : return $\bot$
if $c = C_j$ : return $\bot$
$m' \leftarrow$ E.dec$(k, l_j, c)$
$(m, t) \leftarrow m'$
$passed \leftarrow$ Ovrf$(j, m, t)$
if $\neg passed$ : $m \leftarrow \bot$
return $m$
```

Figure 29: Adversary B, has access to A and oracles Omac and Ovrf. Key space $\mathcal{K}_{enc}$ is the key space from E.enc.

The runtime of B is that of A. For every Oenc query A makes, B computes E.enc once, and calls Omac once. For every Odec query A makes, B computes E.dec once and calls Ovrf once. Note that, alternatively, B could return 0 if *passed* is *true* to avoid having to do E.dec computations. To increase consistency with the other two cases, these computations are still made. We can see that $\Pr[\text{lMAC-PRF}^0_{B,N} = 0] = \Pr[\text{lAE-N3}^0_{A,N} = 0]$ as B perfectly simulates game lAE-N3 with $b = 0$ when its own b is 0. In addition, $\Pr[\text{lMAC-PRF}^1_{B,N} = 0] = \Pr[\text{N3-switch-1}_{A,N} = 0]$ as B perfectly simulates game N3-switch-1 whenever its own b is 1. As a result, we can rewrite our advantage to:

$$\begin{aligned}
\mathbf{Adv}^{\text{lAE}}_{A,N} &= \Pr[\text{lAE-N3}^0_{A,N} = 0] - \Pr[\text{N3-switch-1}_{A,N} = 0] \\
&+ \Pr[\text{N3-switch-1}_{A,N} = 0] - \Pr[\text{lAE-N3}^1_{A,N} = 0] \\
&= \Pr[\text{lMAC-PRF}^0_{B,N} = 0] - \Pr[\text{lMAC-PRF}^1_{B,N} = 0] \\
&+ \Pr[\text{N3-switch-1}_{A,N} = 0] - \Pr[\text{lAE-N3}^1_{A,N} = 0] \\
&= \mathbf{Adv}^{\text{lMAC}}_{B,N} + \Pr[\text{N3-switch-1}_{A,N} = 0] - \Pr[\text{lAE-N3}^1_{A,N} = 0].
\end{aligned}$$

To expand our advantage again, we define game N3-switch-2 in Figure 30. Apart from the Odec query, this game is equivalent to the first switch game. The Odec oracle from N3-switch-2 always returns $\perp$, it is written down more elaborately to include the event *bad*. This is event added to support a well-known proof tactic [6]. Our expanded advantage is:

$$\begin{aligned}
\mathbf{Adv}^{\text{lAE}}_{A,N} &= \mathbf{Adv}^{\text{lMAC}}_{B,N} + \Pr[\text{N3-switch-1}_{A,N} = 0] - \Pr[\text{N3-switch-2}_{A,N} = 0] \\
&+ \Pr[\text{N3-switch-2}_{A,N} = 0] - \Pr[\text{lAE-N3}^1_{A,N} = 0].
\end{aligned}$$

```
Game N3-switch-2_{A,N}
0 :  $L \leftarrow \emptyset$
1 :  $tag \xleftarrow{\$} Func(\mathcal{K}_{mac} \times \mathcal{L} \times \mathcal{M}, \mathcal{T})$
2 :  for $j \in [1..N]$ :
3 :      $k_j \xleftarrow{\$} \mathcal{K}$
4 :      $C_j \leftarrow \perp$
5 :  $b' \leftarrow A$
6 :  return $b'$
```

```
Oracle Oenc(j, l, m)
7 :  if $C_j \neq \perp$ : return $\perp$
8 :  if $l \in L$ : return $\perp$
9 :  $L \leftarrow L \cup \{l\}$
10 : $l_j \leftarrow l$
11 : $(k1, k2) \leftarrow k_j$
12 : $t \leftarrow tag(k2, l_j, m)$
13 : $m' \leftarrow m \| t$
14 : $c \leftarrow$ E.enc$(k1, l_j, m')$
15 : $C_j \leftarrow c$
16 : return $c$
```

```
Oracle Odec(j, c)
17 : if $C_j = \perp$ : return $\perp$
18 : if $c = C_j$ : return $\perp$
19 : $(k1, k2) \leftarrow k_j$
20 : $m' \leftarrow$ E.dec$(k1, l_j, c)$
21 : $(m, t) \leftarrow m'$
22 : $t' \leftarrow tag(k2, l_j, m)$
23 : if $t \neq t'$ : $m \leftarrow \perp$
24 : else :
25 :     $bad \leftarrow true$
26 :     $m \leftarrow \perp$
27 : return $m$
```

Figure 30: N3-switch-2 game, adversary has access to oracles Oenc and Odec. Key space $\mathcal{K}_{mac}$ is the key space from M.mac. Line 24-26 are different compared to N3-switch-1.

As N3-switch-1 and N3-switch-2 are so called identical-until-*bad* [6], meaning they are equivalent as long as the event *bad* is not set to *true*, we know:

$$\Pr[\text{N3-switch-1}_{A,N} = 0] - \Pr[\text{N3-switch-2}_{A,N} = 0] \leq \Pr[bad = true].$$

Below thick lines indicate the technical difference of current proof with the one of N1 in Section 6

As *bad* is set to *true* if, and only if, $t{=}t'$, we can state $\Pr[bad = true] = \Pr[t = t']$. The adversary needs to provide a ciphertext c' that leads to a message m that is used as input to the *tag* function and a tag t. The provided ciphertext may not be the result of the encryption query corresponding to the provided user, which also ensures the message-tag pair derived from this ciphertext cannot be the message-tag pair which is encrypted for the provided user. Because the function *tag* is uniformly random and the output need to match with the newly obtained message-tag pair, the probability that t and t' are equal is $\frac{1}{2^{\mathrm{n}}}$ with every fresh Odec query. Summed over at most Q_d Odec queries we get $\Pr[t = t'] = \Pr[bad = true] \leq \frac{Q_d}{2^{\mathrm{n}}}$ and thus, we can use $\Pr[\text{N3-switch-1}_{A,N} = 0] - \Pr[\text{N3-switch-2}_{A,N} = 0] \leq \Pr[bad = true] \leq \frac{Q_d}{2^{\mathrm{n}}}$ to obtain:

$$\mathbf{Adv}^{\mathrm{lAE}}_{A,N} \leq \mathbf{Adv}^{\mathrm{lMAC}}_{B,N} + \Pr[\text{N3-switch-2}_{A,N} = 0] - \Pr[\text{lAE-N3}^{1}_{A,N} = 0] + \frac{Q_d}{2^{\mathrm{n}}}.$$

We define game N3-switch-3 in Figure 31 to expand our advantage one last time. Switch game 3 is equivalent to switch game 2 but always returns lazily sampled random bits when the outcome of E.enc is valid. It might seem like there is a difference as $t$can no longer become $\perp$. This is not the case as, due to the chosen input spaces of tag, tag can only return $\perp$ whenever c' is already $\perp$. As a result, tag will never influence wether or not c on line 13 is $\perp$. We also simplify Odec as we no longer need the event *bad*. We use this game to expand our advantage to:

$$\begin{aligned}\mathbf{Adv}^{\mathrm{lAE}}_{A,N} \leq \mathbf{Adv}^{\mathrm{lMAC}}_{B,N} &+ \Pr[\text{N3-switch-2}_{A,N} = 0] - \Pr[\text{N3-switch-3}_{A,N} = 0] \\ &+ \Pr[\text{N3-switch-3}_{A,N} = 0] - \Pr[\text{lAE-N3}^{1}_{A,N} = 0] + \frac{Q_d}{2^{\mathrm{n}}}.\end{aligned}$$

```
Game N3-switch-3$_{A,N}$
0 :  $L \leftarrow \emptyset$
1 :  for $j \in [1..N]$ :
2 :      $k_j \xleftarrow{\$} \mathcal{K}_{enc}$
3 :      $C_j \leftarrow \perp$
4 :  $b' \leftarrow A$
5 :  return $b'$

Oracle Oenc$(j, l, m)$
6 :  if $C_j \neq \perp$ : return $\perp$
7 :  if $l \in L$ : return $\perp$
8 :  $L \leftarrow L \cup \{l\}$
9 :  $l_j \leftarrow l$
10 : $t \xleftarrow{\$} \{0,1\}^{\mathrm{n}}$
11 : $m' \leftarrow m \| t$
12 : $c \leftarrow \mathrm{E.enc}(k_j, l, m')$
13 : if $c \neq \perp$ :
14 :     $c \xleftarrow{\$} \{0,1\}^{|c|}$
15 : $C_j \leftarrow c$
16 : return $c$

Oracle Odec$(j, c)$
18 : return $\perp$
```

Figure 31: N3-switch-3 game, adversary has access to oracles Oenc and Odec. Key space $\mathcal{K}_{enc}$ is the key space from E.enc. Line 14 and 15 are different compared to N3-switch-2, and Odec is simplified.

$\Pr[\text{N3-switch-3}_{A,N} = 0]$ and $\Pr[\text{lAE-N3}^{1}_{A,N} = 0]$ are equivalent by definition, giving:

$$\mathbf{Adv}^{\mathrm{lAE}}_{A,N} \leq \mathbf{Adv}^{\mathrm{lMAC}}_{B,N} + \Pr[\text{N3-switch-2}_{A,N} = 0] - \Pr[\text{N3-switch-3}_{A,N} = 0] + \frac{Q_d}{2^{\mathrm{n}}}.$$

Next, we can rewrite $\Pr[\text{N3-switch-2}_{A,N} = 0] - \Pr[\text{N3-switch-3}_{A,N} = 0]$ into a lE advantage. To do so, we define adversary C against lE in Figure 32. This adversary is playing game lE-IND-\$-CPA (Figure 9), and has access to A.

```
Adverary $C$
0 :  $L \leftarrow \emptyset$
1 :  for $j \in [1..N]$ :
2 :      $C_j \leftarrow \perp$
3 :  $b' \leftarrow$ run $A$
4 :  return $b'$

if $A$ calls Oracle Oenc$(j, l, m)$
5 :  if $C_j \neq \perp$ : return $\perp$
6 :  if $l \in L$ : return $\perp$
7 :  $L \leftarrow L \cup \{l\}$
8 :  $l_j \leftarrow l$
9 :  $t \xleftarrow{\$} \{0,1\}^{\mathrm{n}}$
10 : $m' \leftarrow m \| t$
11 : $c \leftarrow \mathrm{Oenc}(j, l, m')$
12 : $C_j \leftarrow c$
13 : return $c$

if $A$ calls Oracle Odec$(j, c)$
14 : return $\perp$
```

Figure 32: Adversary C, has access to A and oracle Oenc. Note the Oenc in line 11 refers to the encryption oracle Oenc that C has access to, not the oracle Oenc A has access to.

The runtime of C is that of A. For every Oenc query A makes, C makes one Oenc query. We can see that $\Pr[\text{N3-switch-2}_{A,N} = 0] = \Pr[\text{lE-IND-\$-CPA}^0_{C,N} = 0]$ as C perfectly simulates N3-switch-2 when its own b is 0. When its own b is 1, C perfectly simulates N3-switch-3 giving $\Pr[\text{N3-switch-3}_{A,N} = 0] = \Pr[\text{lE-IND-\$-CPA}^1_{C,N} = 0]$. This leads to:

$$\begin{aligned}\mathbf{Adv}^{\text{lAE}}_{A,N} &\leq \mathbf{Adv}^{\text{lMAC}}_{B,N} + \Pr[\text{N3-switch-2}_{A,N} = 0] - \Pr[\text{N3-switch-3}_{A,N} = 0] + \frac{Q_d}{2^{\mathrm{n}}} \\ &\leq \mathbf{Adv}^{\text{lMAC}}_{B,N} + \Pr[\text{lE-IND-\$-CPA}^0_{C,N} = 0] - \Pr[\text{lE-IND-\$-CPA}^1_{C,N} = 0] + \frac{Q_d}{2^{\mathrm{n}}} \\ &\leq \mathbf{Adv}^{\text{lMAC}}_{B,N} + \mathbf{Adv}^{\text{lE}}_{C,N} + \frac{Q_d}{2^{\mathrm{n}}}.\end{aligned}$$

Thus Proving:

$$\mathbf{Adv}^{\text{lAE}}_{A,N} \leq \mathbf{Adv}^{\text{lMAC}}_{B,N} + \mathbf{Adv}^{\text{lE}}_{C,N} + \frac{Q_d}{2^{\mathrm{n}}}. \qquad \square$$