Bachelor Thesis

Radboud Honours Academy

# TBD

*Author:*
Stijn Vandenput

*Supervisors:*
Martijn Stam
Bart Mennink

20/05/2022

# Abstract

# Contents

# 1 Introduction

should consist of:

- explaining the challenge
- my contribution

Within the field of cryptography there are a two different fields, symmetric and asymmetric crypto. The former is about situations where the users have a shared secret already, the latter is about situations where this is not the case. Sometimes work in asymmetric crypto uses constructions that are more common in symmetric crypto, and in this fashion, a paper by Giacon, Kiltz and Poettering [1], which we henceforth call GKP, uses a construction that is very similar to Authenticated Encryption following the generic encrypt-then-MAC construction from Bellare and Namprempre [2]. This construction has since been revised in a paper by Namprempre, Rogaway and Thomas Shrimpton [3], which we henceforth call NRS, to be better applicable to common use cases. The aim of this thesis is to apply the knowledge of NRS to the setting of GKP and in doing so, create a primitive for authenticated encryption in a asymmetric crypto setting.

# 2 Preliminaries

should consist of:

- general notation
- AE
- PKE schemes?
- nonces vs locks (and different iv's)
- game based security notions
- ind$/ind-cpa/ind-cca

## 2.1 general notation

Below is a table which highlights the differences in notation between GKPand NRS, as well as give the notation I will be using.

| Name | GKP | NRS | my notation | rough meaning |
|---|---|---|---|---|
| message space | $\mathcal{M}$ | $\mathcal{M}$ | $\mathcal{M}$ | set of all possible message options |
| message | $m$ | $M$ | $m$ | message the user sends |
| ciphertext space | $\mathcal{C}$ | - | $\mathcal{C}$ | set of all possible ciphertext options |
| ciphertext | $c$ | $C$ | $c$ | encrypted message |
| associated data space | - | $\mathcal{A}$ | $\mathcal{A}$ | set of all possible associated data options |
| associated data | - | $A$ | $a$ | data you want to authenticate but not encrypt |
| tag space | $\mathcal{C}$ | - | $\mathcal{T}$ | set off all possible tag options |
| tag | $c$ | $T$ | $t$ | output of mac function |
| key space | $\mathcal{K}$ | $\mathcal{K}$ | $\mathcal{K}$ | set of all possible key options |
| key | $k$ | $K$ | $k$ | user key |
| nonce space | - | $\mathcal{N}$ | $\mathcal{N}$ | set of all nonce options |
| nonce | - | $n$ | $n$ | number only used once |
| lock space | $\mathcal{T}$ | - | $\mathcal{L}$ | set of all possible lock options |
| lock | $t$ | - | $l$ | nonce that is bound to the user |
| users | $N$ | - | $N$ | number of users |
| adversary | A | $\mathcal{A}$ | $A$ | the bad guy |
| random sampling | $\xleftarrow{\$}$ | $\twoheadleftarrow$ | $\xleftarrow{\$}$ | get a random ellermetn form the set |
| result of randomised function | $\xleftarrow{\$}$ | - | $\leftarrow$ | get the result of a randomised function with given inputs |
| result of function | $\leftarrow$ | $\leftarrow$ | $\leftarrow$ | get the result of a function with given inputs |
| concatination | a\|\|b | a\|\|b | a\|\|b | concatanation of two strings |
| true | *true* | - | *true* | boolean true |
| false | *false* | - | *false* | boolean false |
| invalid | $\perp$ | $\perp$ | $\perp$ | operation is invalid |

strings are binary and bit-wise

## 2.2 AE

## 2.3 PKE schemes?

## 2.4 nonces vs locks (and different iv's)

## 2.5 game based security notions

## 2.6 ind-$/ind-lor/ind-cpa/ind-cca

also note active and passive attackers

# 3 GKP and NRS in more detail

In this section we will note down the specific details that are on interest for us, from both GKP and NRS. Some notations will be different from the original paper in order to make it more consistent with each other, as well as our own notation. This will make it easier to see the differences and similarities between the two. whenever necessary, these changes will be elaborated upon.

## 3.1 Existing notion from GKP

The part of GKP that is of interest for us is the ADEM-then-AMAC construction. They start with a augmented data encapsulation mechanism, ADEM for short, that is secure against passive attackers. They combine this existing scheme with a augmented message authentication code, AMAC for short, to create a ADEM' that is secure against active attackers. The details of this are found below.

### 3.1.1 notation

$\mathcal{M}$ is a message space, $\mathcal{K}$ is a finite key space, $\mathcal{L}$ is a lock space and $\mathcal{C}$ is a ciphertext space. $N$ is the number of users. The locks are called tags in GKP, additionally, we call the output of the AMAC the tag instead of the ciphertext.

### 3.1.2 used primitives

ADEM: the ADEM exists of tuple (A.enc, A.dec), A.enc is a deterministic algorithm that 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 $\mathcal{C}$. A.dec is a deterministic algorithm that 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 $\perp$ to indicate rejection. The correctness requirement is that for every combination of $k$, $l$ and $m$ we have $\mathrm{A.dec}(k, l, \mathrm{A.enc}(k, l, m)) = m$. The security of the ADEM is defined with $\mathbf{Adv}^{\text{l-ind-cpa}}_{\text{ADEM},A,N} = |\Pr[\text{L-IND-CPA}^0_{A,N} = 1] - \Pr[\text{L-IND-CPA}^1_{A,N} = 1]|$, defined by the following game:

| **Game** $\text{L-IND-CPA}^b_{A,N}$ | **Oracle** $\mathrm{Oenc}(j, l, m_0, m_1)$ |
|---|---|
| $L \leftarrow \emptyset$ | **if** $C_j \neq \emptyset$ : **return** $\perp$ |
| **for** $j \in [1..N]$ : | **if** $l \in L$ : **return** $\perp$ |
| $\quad k_j \xleftarrow{\$} \mathcal{K}$ | $L \leftarrow L \cup \{l\}$ |
| $\quad C_j \leftarrow \emptyset$ | $l_j \leftarrow l$ |
| $b' \leftarrow A$ | $c \leftarrow \mathrm{A.enc}(k_j, l_j, m_b)$ |
| **return** $b'$ | $C_j \leftarrow C_j \cup \{c\}$ |
| | **return** $c$ |

Figure 1: L-IND-CPA game, $A$ has access to oracle Oenc. The corresponding game can be found in GKP figure 9 (note that this has a decryption oracle the ADEM is not allowed to use).

AMAC: the AMAC exists of tuple (M.mac, M.vrf). M.mac is a deterministic algorithm that takes a key $k$ in $\mathcal{K}$, a lock $l$ in $\mathcal{L}$, and a message $m$ in $\mathcal{M}$ and outputs a ciphertext $t$ in $\mathcal{T}$. M.vrf takes a key $k$ in $\mathcal{K}$, a lock $l$ in $\mathcal{L}$, a message $m$ in $\mathcal{M}$ and a ciphertext $t$ 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*. The security of the AMAC is defined with $\mathbf{Adv}^{\text{L-MIOT-UF}}_{\text{AMAC},A,N} = \Pr[\text{L-MIOT-UF}_{A,N} = 1]$, defined by the following game:

| **Game** L-MIOT-UF$_{A,N}$ | **Oracle** Omac$(j, l, m)$ | **Oracle** Ovrf$(j, m, t)$ |
|---|---|---|
| 0 : forged $\leftarrow 0$ | 7 : **if** $T_j \neq \emptyset$ : **return** $\perp$ | 14 : **if** $T_j = \emptyset$ : **return** $\perp$ |
| 1 : $L \leftarrow \emptyset$ | 8 : **if** $l \in L$ : **return** $\perp$ | 15 : **if** $(m, t) \in T_j$ : **return** $\perp$ |
| 2 : **for** $j \in [1..N]$ : | 9 : $L \leftarrow L \cup \{l\}$ | 16 : **if** M.vrf$(k_j, l_j, m, t)$ : |
| 3 : $\quad k_j \leftarrow^{\$} \mathcal{K}$ | 10 : $l_j \leftarrow l$ | 17 : $\quad$ forged $\leftarrow 1$ |
| 4 : $\quad T_j \leftarrow \emptyset$ | 11 : $t \leftarrow$ M.mac$(k_j, l_j, m)$ | 18 : $\quad$ **return** *true* |
| 5 : **run** $A$ | 12 : $T_j \leftarrow T_j \cup \{(m, t)\}$ | 19 : **else** : **return** *false* |
| 6 : **return** forged | 13 : **return** $t$ | |

Figure 2: L-MIOT-UF game, $A$ has access to oracles Omac and Ovrf and the locks in line 11 and 16 are the same. The corresponding game can be found in GKP figure 15.

#### 3.1.3 goal

The goal is to create a ADEM' the ADEM' exists of tuple (A.enc', A.dec'), A.enc' is a deterministic algorithm that 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 $\mathcal{C}$. A.dec' is a deterministic algorithm that 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 $\perp$ to indicate rejection. The correctness requirement is that for every combination of $k$, $l$ and $m$ we have A.dec'$(k, l, \text{A.enc'}(k, l, m)) = m$.

#### 3.1.4 Security model

The security of the ADEM' is defined with $\mathbf{Adv}^{\text{l-ind-cca}}_{\text{ADEM'},A,N} = |\Pr[\text{L-IND-CCA}^0_{A,N} = 1] - \Pr[\text{L-IND-CCA}^1_{A,N} = 1]|$, defined by the following game:

| **Game** L-IND-CCA$^b_{A,N}$ | **Oracle** Oenc$(j, l, m_0, m_1)$ | **Oracle** Odec$(j, c)$ |
|---|---|---|
| 0 : $L \leftarrow \emptyset$ | 6 : **if** $C_j \neq \emptyset$ : **return** $\perp$ | 13 : **if** $C_j = \emptyset$ : **return** $\perp$ |
| 1 : **for** $j \in [1..N]$ : | 7 : **if** $l \in L$ : **return** $\perp$ | 14 : **if** $c \in C_j$ : **return** $\perp$ |
| 2 : $\quad k_j \leftarrow^{\$} \mathcal{K}$ | 8 : $L \leftarrow L \cup \{l\}$ | 15 : $m \leftarrow$ A.dec'$(k_j, l_j, c)$ |
| 3 : $\quad C_j \leftarrow \emptyset$ | 9 : $l_j \leftarrow l$ | 16 : **return** $m$ |
| 4 : $b' \leftarrow A$ | 10 : $c \leftarrow$ A.enc'$(k_j, l_j, m_b)$ | |
| 5 : **return** $b'$ | 11 : $C_j \leftarrow C_j \cup \{c\}$ | |
| | 12 : **return** $c$ | |

Figure 3: L-IND-CCA game, $A$ has access to oracles Oenc and Odec and the locks in line 10 and 15 are the same. The corresponding game can be found in GKP figure 9.

### 3.1.5 construction

The goal is met by creating A.enc' and A.dec' calls which are build using the calls the primitives provide us:

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

Figure 4: A.enc' and A.dec' calls, The corresponding calls can be found in GKP figure 16.

The construction is deemed secure as for any $N$ and a $A$ that makes $Q_d$ many Odec queries, the exist $B$ and $C$ such that $\mathbf{Adv}^{\text{l-ind-cca}}_{\text{ADEM'},A,N} \leq 2\mathbf{Adv}^{\text{l-miot-uf}}_{\text{AMAC},B,N} + \mathbf{Adv}^{\text{l-ind-cpa}}_{\text{ADAM},C,N}$ holds. Where the running time of $B$ is at most that of $A$ plus the time required to run $N$-many ADEM encapsulations and $Q_d$-many ADEM decapsulations and the running time of $C$ is the same as the running time of $A$. Additionally, $B$ poses at most $Q_d$-many Ovrf queries, and $C$ poses no Odec query.

## 3.2 Existing notation from NRS

Below are the security notations from NRS that are relevant for this thesis. Some of the variable names are rewritten towards a common notation with GKP.

### 3.2.1 notation

$\mathcal{K}$ is a nonempty key space, $\mathcal{N}$ is a non-empty nonce space, $\mathcal{M}$ is a message space and $\mathcal{A}$ is the associated-data space. $\mathcal{M}$ contains at least two strings, and if $\mathcal{M}$ and $\mathcal{A}$ contain a string of length x, they must contain all strings of length x.

### 3.2.2 used primitives

- nE: A nonce-based encryption scheme is defined by triple $\Pi = (\mathcal{K}, \mathrm{E}, \mathrm{D})$. E is a deterministic encryption algorithm that takes three inputs $(k, n, m)$ to 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 $\perp$. D is the decryption algorithm that takes three inputs $(k, n, c)$ to a value $m$. E and D are inverse of each other implying 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 $\mathrm{E}(k, n, m) = c$). The security is defined as $\mathbf{Adv}^{\text{nE}}_{\Pi,A} = |\Pr[\text{nE-IND\$}^0_A = 1] - \Pr[\text{nE-IND\$}^1_A = 1]|$, where nE-IND\$ is defined as follows:

| **Game** nE-IND$\$^b_A$ | **Oracle** $\mathrm{Oenc}(n, m)$ |
|---|---|
| $U \leftarrow \emptyset$ | **if** $n \in U$ : **return** $\perp$ |
| $k \leftarrow^{\$} \mathcal{K}$ | $U \leftarrow U \cup \{n\}$ |
| $b' \leftarrow A$ | $c \leftarrow \mathrm{E}(k, n, m)$ |
| **return** $b'$ | **if** $b = 1 \wedge c \neq \perp$ : |
| | $\quad c \leftarrow^{\$} \{0,1\}^{\lvert c \rvert}$ |
| | **return** $c$ |

Figure 5: nE-IND\$ game, $A$ has access to oracle Oenc and $U$ is the set of used nonces

- MAC: The MAC is a deterministic algorithm F that takes in a $k$ in $\mathcal{K}$ and a string $m$ and outputs either a n-bit length $t$ or $\perp$. The domain of F is the set X such that $\mathrm{F}(k, m) \neq \perp$. This domain may not depend on $k$. The security is defined as $\mathbf{Adv}^{\mathrm{MAC}}_{\mathrm{F},A} = |\Pr[\text{MAC-PRF}^0_A = 1] - \Pr[\text{MAC-PRF}^1_A = 1]|$, where MAC-PRF is defined as follows:

| **Game** MAC-PRF$^b_A$ | **Oracle** $\mathrm{Omac}(m)$ |
|---|---|
| $U \leftarrow \emptyset$ | **if** $m \in U$ : **return** $\perp$ |
| $k \leftarrow^{\$} \mathcal{K}$ | $U \leftarrow U \cup \{m\}$ |
| $b' \leftarrow A$ | $t \leftarrow \mathrm{F}(k, m)$ |
| **return** $b'$ | **if** $b = 1 \wedge t \neq \perp$ : |
| | $\quad t \leftarrow^{\$} \{0,1\}^{\lvert t \rvert}$ |
| | **return** $t$ |

Figure 6: MAC-PRF, $A$ has access to oracle Omac and $U$ is the set of used messages

#### 3.2.3 goal

The goal is a nonce-based authenticated encryption scheme defined by triple $\mathit{\Pi} = (\mathcal{K}, \mathrm{E}, \mathrm{D})$. E is a deterministic encryption algorithm that takes four inputs $(k, n, a, m)$ to 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 $\perp$. D is the decryption algorithm that takes four inputs $(k, n, a, c)$ to a value $m$. E and D are inverse of each other implying 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 $\mathrm{E}(k, n, a, m) = c$)

#### 3.2.4 Security model

The security is defined as $\mathbf{Adv}^{\mathrm{nAE}}_{\mathit{\Pi},A} = |\Pr[\text{nAE-IND\$}^0_A = 1] - \Pr[\text{nAE-IND\$}^1_A = 1]|$, where nAE-IND\$ is defined as follows:

| **Game** nAE-IND$$^b_A$ | **Oracle** $\mathrm{Oenc}(n, a, m)$ | **Oracle** $\mathrm{Odec}(n, a, c)$ |
|---|---|---|
| $U \leftarrow \emptyset$ | **if** $n \in U$ : **return** $\perp$ | **if** $b = 1$ : **return** $\perp$ |
| $Q \leftarrow \emptyset$ | $U \leftarrow U \cup \{n\}$ | **if** $(n, a, \_, c) \in Q$ : **return** $\perp$ |
| $k \leftarrow^{\$} \mathcal{K}$ | **if** $(n, a, m, \_) \in Q$ : **return** $\perp$ | $m \leftarrow \mathrm{D}(k, n, a, c)$ |
| $b' \leftarrow A$ | $c \leftarrow \mathrm{E}(k, n, a, m)$ | $Q \leftarrow Q \cup \{(n, a, m, c)\}$ |
| **return** $b'$ | **if** $b = 1 \wedge c \neq \perp$ : | **return** $m$ |
| | $\quad c \leftarrow^{\$} \{0,1\}^{\lvert c \rvert}$ | |
| | $Q \leftarrow Q \cup \{(n, a, m, c)\}$ | |
| | **return** $c$ | |

Figure 7: nAE-IND$ game, $A$ has access to oracles Oenc and Odec, $U$ is the set of used nonces and $Q$ is the set of query results. _ denotes a variable that is irrelevant

#### 3.2.5 construction

The goal is met by creating different schemes that combine the mac and nE into a nAE. We define the constructions secure as 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 the can be viewed in NRS figure 6.

# 4 New Definition

should consist of:

- syntax of the primitive (input, output, correctness, tidiness, expected bounds)
- game based code
- explanation of the choices made

## 4.1 notation

$\mathcal{K}$ is a non-empty key space, $\mathcal{L}$ is a non-empty lock space and $\mathcal{M}$ is a message space. $\mathcal{M}$ contains at least two strings, and if it contains a string of length x, it must contain all strings of length x. $N$ is the number of users.

## 4.2 goal

The end goal is to build a lock-based one time use Authenticated Encryption scheme (loAE). The AE scheme is defined by tuple (AE.enc, AE.dec). AE.enc is a deterministic encryption algorithm that takes three inputs $(k, l, m)$ to 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 $\perp$. AE.dec is the decryption algorithm that takes three inputs $(k, n, c)$ to a value $m$. AE.enc and EA.dec are required to have correctness (if AE.enc$(k, l, m) = c \neq \perp$, then AE.dec$(k, l, c) = m$) and tidiness (if AE.dec$(k, l, c)$ $= m \neq \perp$, then AE.enc$(k, l, m) = c$).

### 4.3 Security model

The security is defined as $\mathbf{Adv}^{\mathrm{loAE}}_{A,N} = |\Pr[\text{loAE-IND\$}^0_{A,N} = 1] - \Pr[\text{loAE-IND\$}^1_{A,N} = 1]|$, where loAE-IND$ is defined as follows:

| **Game** loAE-IND$\$^b_{A,N}$ | **Oracle** $\mathrm{Oenc}(j, l, m)$ | **Oracle** $\mathrm{Odec}(j, c)$ |
|---|---|---|
| $L \leftarrow \emptyset$ | **if** $C_j \neq \perp$ : **return** $\perp$ | **if** $b = 1$ : **return** $\perp$ |
| **for** $j \in [1..N]$ : | **if** $l \in L$ : **return** $\perp$ | **if** $C_j = \perp$ : **return** $\perp$ |
| $\quad k_j \xleftarrow{\$} \mathcal{K}$ | $L \leftarrow L \cup \{l\}$ | **if** $c = C_j$ : **return** $\perp$ |
| $\quad C_j \leftarrow \perp$ | $l_j \leftarrow l$ | $m \leftarrow \mathrm{AE.dec}(k_j, l_j, c)$ |
| $b' \leftarrow A$ | $c \leftarrow \mathrm{AE.enc}(k_j, l_j, m)$ | **return** $m$ |
| **return** $b'$ | **if** $b = 1 \wedge c \neq \perp$ : | |
| | $\quad c \xleftarrow{\$} \{0,1\}^{\lvert c \rvert}$ | |
| | $C_j \leftarrow c$ | |
| | **return** $c$ | |

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

# 5 Constructions

should consist of:

- what old primitives to use
- the old primitives
- how to construct the new primitive from old primitives
- security bounds + proof
- formal comparison with other choices

### 5.1 used primitives

$\mathcal{K}$ is a nonempty key space, $\mathcal{N}$ is a non-empty nonce space and $\mathcal{M}$ is a message space. $\mathcal{M}$ contain at least two strings, and if it contain a string of length x, it must contain all strings of length x. $N$ is the number of users.

- loDEM: a lock-based one time use DEM scheme is defined by tuple (E.enc, E.dec). E.enc is a deterministic encryption algorithm that takes three inputs $(k, l, m)$ to 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 $\perp$. E.dec is the decryption algorithm that takes three inputs $(k, l, c)$ to a value $m$. E.enc and E.dec are required to have correctness (if $\mathrm{E.enc}(k, l, m) = c \neq \perp$, then $\mathrm{E.dec}(k, l, c) = m$) and tidiness (if $\mathrm{E.dec}(k, l, c) = m \neq \perp$, then $\mathrm{E.enc}(k, l, m) = c$). The security is defined as $\mathbf{Adv}^{\mathrm{loDEM}}_{A,N} = |\Pr[\text{loDEM-IND\$}^0_{A,N} = 1] - \Pr[\text{loDEM-IND\$}^1_{A,N} = 1]|$, where loDEM-IND$ is defined as follows:

**Game** $\text{loDEM-IND\$}^{b}_{A,N}$

$L \leftarrow \emptyset$
**for** $j \in [1..N]$ :
  $k_j \leftarrow^{\$} \mathcal{K}$
  $C_j \leftarrow \bot$
$b' \leftarrow A$
**return** $b'$

**Oracle** $\text{Oenc}(j, l, m)$

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

Figure 9: loDEM-IND$

- loMAC: The lock-based one time use MAC is a deterministic algorithm M.mac that takes in 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 $t$, or $\bot$. If, and only if, $(k, l, m)$ is not in $\mathcal{K} \times \mathcal{L} \times \mathcal{M}$, $t$ will be $\bot$. the security is defined as $\mathbf{Adv}^{\text{loMAC}}_{\text{F},A,N} = |\Pr[\text{loMAC-PRF}^{0}_{A,N} = 1] - \Pr[\text{loMAC-PRF}^{1}_{A,N} = 1]|$, where loMAC-PRF is defined as follows:

**Game** $\text{loMAC-PRF}^{b}_{A,N}$

$L \leftarrow \emptyset$
**for** $j \in [1..N]$ :
  $k_j \leftarrow^{\$} \mathcal{K}$
  $T_j \leftarrow \bot$
$b' \leftarrow A$
**return** $b'$

**Oracle** $\text{Omac}(j, l, m)$

**if** $T_j \neq \bot$ : **return** $\bot$
**if** $l \in L$ : **return** $\bot$
$L \leftarrow L \cup \{l\}$
$l_j \leftarrow l$
$t \leftarrow \text{M.mac}(k_j, l_j, m)$
**if** $b = 1 \wedge t \neq \bot$ :
  $t \leftarrow^{\$} \{0,1\}^{|t|}$
$T_j \leftarrow t$
**return** $t$

**Oracle** $\text{Ovrf}(j, m, t)$

**if** $T_j = \bot$ : **return** $\bot$
**if** $(m, t) = T_j$ : **return** $\bot$
**if** $b = 1$ : **return** *false*
$t' \leftarrow \text{M.mac}(k_j, l_j, m)$
**if** $t = t'$
  **return** *true*
**return** *false*

Figure 10: loMAC-PRF, $A$ has access to oracle Omac

**Game** loMAC-PRF$^b_{A,N}$

$L \leftarrow \emptyset$
**for** $j \in [1..N]$ :
  $k_j \leftarrow^{\$} \mathcal{K}$
  $T_j \leftarrow \perp$
  $F_j$ **init**
$b' \leftarrow A$
**return** $b'$

**Oracle** Omac$(j, l, m)$

**if** $T_j \neq \perp$ : **return** $\perp$
**if** $l \in L$ : **return** $\perp$
$L \leftarrow L \cup \{l\}$
$l_j \leftarrow l$
$t \leftarrow \text{M.mac}(k_j, l_j, m)$
**if** $b = 1 \wedge t \neq \perp$ :
  $t \leftarrow F_j(k_j, l_j, m)$
$T_j \leftarrow t$
**return** $t$

**Oracle** Ovrf$(j, m, t)$

**if** $T_j = \perp$ : **return** $\perp$
**if** $(m, t) = T_j$ : **return** $\perp$
**if** $b = 0$ : $t' \leftarrow \text{M.mac}(k_j, l_j, m)$
**if** $b = 1$ : $t' \leftarrow F_j(k_j, l_j, m)$
**if** $t = t'$
  **return** *true*
**return** *false*

Figure 11: loMAC-PRF v2, $A$ has access to oracle Omac **init** selects a uniformly random function from $\mathcal{K} \times \mathcal{L} \times \mathcal{M}$ to the set of all n-bit long strings

I am not sure which of these figures is better, figure 9 is closes to the loAE game, but get the feeling it might not be fully accurate as a prf is likely to return *true*when enough guesses are made as the output is always n-bits. If you can make n$^2$ queries, you can distinguish between b = 0 and b = 1 as a message will for sure verify correctly once if you try to verify it will all possible tags, where this strategy would not work when distinguishing with a prf as a prf will also verify correctly once if you try to verify it will all possible tags. However, I think we can disregard this as this problem would otherwise also arise when defining the loAE security. The second one feels more accurate to me as it also matches better with the mac requirement from NRSon pdf page 8. I think figure 9 is good for this setting but figure 10 might be a bit more complete so I am not sure which to use

## 5.2 construction

The loAE schemes we will look at are constructed from the loDEM and the loMAC. Following NRS, three ways to construct this loAE are of interest, namely the ones following from the N1, N2 and N3 scheme. One thing to keep in mind with this that these schemes would originally use associated data. For now we can discard this but it is not proven that the same security results would also follow from this case without associated data. Down here the schemes, adjusted to our setting, can be found, followed by the AE.enc and AE.dec calls that can we construct following these schemes. These calls can be plugged into game loAE-IND$ to find their respective security games.

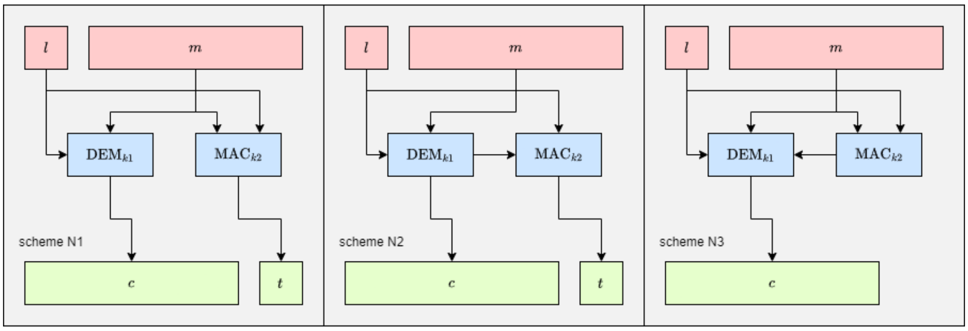


Figure 12: Adjusted N schemes from NRS

| $\text{AE.enc}(k, l, m)$ | $\text{AE.dec}(k, l, c)$ |
|---|---|
| $(k1, k2) \leftarrow k$ | $(k1, k2) \leftarrow k$ |
| $c' \leftarrow \text{E.enc}(k1, l, m)$ | $(c', t) \leftarrow c$ |
| $t \leftarrow \text{M.mac}(k2, l, m)$ | $m \leftarrow \text{E.dec}(k1, l, c')$ |
| $c \leftarrow (c', t)$ | $t' \leftarrow \text{M.mac}(k2, l, m)$ |
| $\textbf{return } c$ | $\textbf{if } t = t' : \textbf{return } m$ |
| | $\textbf{else } : \textbf{return } \perp$ |

Figure 13: Calls based on N1

| $\text{AE.enc}(k, l, m)$ | $\text{AE.dec}(k, l, c)$ |
|---|---|
| $(k1, k2) \leftarrow k$ | $(k1, k2) \leftarrow k$ |
| $c' \leftarrow \text{E.enc}(k1, l, m)$ | $(c', t) \leftarrow c$ |
| $t \leftarrow \text{M.mac}(k2, l, c')$ | $m \leftarrow \text{E.dec}(k1, l, c')$ |
| $c \leftarrow (c', t)$ | $t' \leftarrow \text{M.mac}(k2, l, c')$ |
| $\textbf{return } c$ | $\textbf{if } t = t' : \textbf{return } m$ |
| | $\textbf{else } : \textbf{return } \perp$ |

Figure 14: Calls based on N2

| $\text{AE.enc}(k, l, m)$ | $\text{AE.dec}(k, l, c)$ |
|---|---|
| $(k1, k2) \leftarrow k$ | $(k1, k2) \leftarrow k$ |
| $t \leftarrow \text{M.mac}(k2, l, m)$ | $m' \leftarrow \text{E.dec}(k1, l, c)$ |
| $m' \leftarrow m\|t$ | $(m, t) \leftarrow m'$ |
| $c \leftarrow E.enc(k1, l, m')$ | $t' \leftarrow \text{M.mac}(k2, l, m)$ |
| **return** $c$ | **if** $t = t'$ : **return** $m$ |
| | **else** : **return** $\perp$ |

Figure 15: Calls based on N3

We define the constructions secure if there is a tight reduction from breaking the loAE-security of the scheme to breaking the loDEM-security and the loMAC security of the underlying primitives.

### 5.3 Security Bounds

### 5.4 Comparison with existing alternatives

## 6 Use cases

should consist of:

- possible use cases

### 6.1 PKE schemes

## 7 Related Work

**Location not final yet**

## 8 Conclusion

# 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] 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.

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

# 9 Appendix