Bachelor Thesis

Radboud University

TBD

Author:
Stijn Vandenput

Supervisors:
Martijn Stam
Bart Mennink

20/05/2022

Abstract

Contents

1 Introduction

Although symmetric and asymmetric cryptography are both subfields of cryptography, their research area's can be quite separated. This can lead to knowledge gaps between the two when work in asymmetric crypto uses constructions that are more common in symmetric crypto or the other way around. 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. In this revision, a new set of constructions is given that be better applicable to common use cases. The aim of this thesis is to apply the knowledge from NRS to the setting of GKP and while doing so, create a new primitive for authenticated encryption suited for asymmetric settings.

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 $\|$ y, a being the result of b as a $\leftarrow$ b and taking a random sampling from y and assigning it to x as x $\xleftarrow{\$}$ y. We allow a single type of error message written as $\perp$. $\mathcal{K}$ is a nonempty key space, $\mathcal{L}$ is a lock space, $\mathcal{N}$ is a nonce space, $\mathcal{M}$ is a message space and $\mathcal{A}$ is the associated-data space. $\mathcal{M}$ contain at least two strings, and if $\mathcal{M}$ or $\mathcal{A}$ contain a string of length x, it must contain all strings of length x. N is the number of users.

2.2 AE

2.3 PKE Schemes?

2.4 Nonces vs Locks (and different iv's)

2.5 Game Based Security Notions

2.6 Security Notions

ind-$/ind-lor/ind-cpa/ind-cca also note active and passive attackers

2.7 Security Proofs

3 NRS and GKP in Detail

In this section we explain the parts from GKP and NRS important to our work. Some notations will be different from the original papers for improved consistency. What are called tags in GKP, we will call locks instead to avoid confusion with the output of mac and we call the output of the AMAC the tag instead of the ciphertext. The security notions from NRS are written in a game-based format (**todo citation not yet in crypto.bib**) in order to better match the notation from GKP and be more adaptable to a multi-user setting. afterwards, a comparison is made between the two.

3.1 NRS

A paper written by Bellare and Namprempre [2] is reconsidered in NRS. This paper discusses 3 generic ways to construct a authenticated encryption scheme: encrypt-then-mac, encrypt-and-mac and mac-then-encrypt. In this paper, encrypt-then-mac is considered the only secure one when using probabilistic encryption as a building block. Within NRS these constructions are generalized to using nonce- or iv-based encryption as a building block to create nonce-based authenticated encryption schemes, nAEs for short. For this thesis we will look at the constructions using a nonce-based encryption, nE for short, and a PRF secure MAC function.

3.1.1 Used Primitives

nE: A nonce-based encryption scheme is defined by 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 $\mathrm{E}(k, n, m) = c$). The adversary A is not allowed to repeat nonces. The security is defined as $\mathbf{Adv}^{\mathrm{nE}}_{\mathit{\Pi},A} = |\Pr[\text{nE-IND\$}^0_A = 1] - \Pr[\text{nE-IND\$}^1_A = 1]|$, where nE-IND$ is in figure 1.

Game nE-IND$\$^b_A$	**Oracle** $\mathrm{Oenc}(n, m)$
$0: \quad U \leftarrow \emptyset$	$5: \quad \textbf{if } n \in U : \textbf{return } \perp$
$1: \quad k \xleftarrow{\$} \mathcal{K}$	$6: \quad U \leftarrow U \cup \{n\}$
$2: \quad b' \leftarrow A$	$7: \quad c \leftarrow \mathrm{E}(k, n, m)$
$3: \quad \textbf{return } b'$	$8: \quad \textbf{if } b = 1 \wedge c \neq \perp :$
	$9: \quad\quad c \xleftarrow{\$} \{0,1\}^{\lvert c \rvert}$
	$10: \quad \textbf{return } c$

Figure 1: nE-IND$ game, A has access to oracle Oenc and U is the set of used nonces.

MAC: A MAC scheme is defined by algorithm F that takes a key k in $\mathcal{K}$ and a string m and outputs either a n-bit length tag t or $\perp$. The domain of F is the set X such that all x = $\mathrm{F}(k, m) \neq \perp$ is in X, 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 in figure 2.

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

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

3.1.2 Nonce-Based Authenticated Encryption

A nonce-based authenticated encryption scheme is defined by triple $\Pi = (\mathcal{K}, \mathrm{E}, \mathrm{D})$. 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 $\bot$. 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 $\mathrm{E}(k, n, a, m) = c \neq \bot$, then $\mathrm{D}(k, n, a, c) = m$) and tidiness (if $\mathrm{D}(k, n, a, c) = m \neq \bot$, then $\mathrm{E}(k, n, a, m) = c$). The adversary A is not allowed to repeat nonces. The security is defined as $\mathbf{Adv}^{\mathrm{nAE}}_{\Pi,A} = |\Pr[\text{nAE-IND\$}^0_A = 1] - \Pr[\text{nAE-IND\$}^1_A = 1]|$, where nAE-IND$ is in figure 3.

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

Figure 3: 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. Decryption queries are added to Q, in contrast to GKP, as encryption is still allowed after decryption.

3.1.3 Construction

A nAE scheme is constructed by several 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 they can be viewed in figure 6 of NRS. Noteworthy is that these relate to encrypt-and-mac, encrypt-then-mac,and mac-then-encrypt respectively, showing the general believe that encrypt-then-mac is the only safe construction does not transfer to this setting.

3.2 GKP

In GKP, the concept of augmentation using locks is introduced. The authors start by showing some data encapsulation mechanisms are vulnerable to a passive multi-instance distinguishing- and key recovery attack in a one time use setting. The define the augmented data encapsulation mechanisms, ADEM for short, that uses locks to negate these insecurities. They follow by discussing how a ADEM that is secure against passive attacks can be combined with a MAC that is augmented in a similar fashion, called a AMAC, to construct ADEM' that is safe against active attackers. This construction is similar to construction N2 from NRS. In GKP $\mathcal{M}$ is not required 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.

3.2.1 Used Primitives

ADEM: A ADEM scheme is defined by tuple $(\mathrm{A.enc}, \mathrm{A.dec})$. 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 $\perp$ 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 user is only allowed one encryption query and locks may not repeat between users. Decryption queries are only allowed after the encryption. The security of the ADEM is defined as $\mathbf{Adv}^{\text{l-ind-cpa}}_{\text{ADEM},A,N} = |\Pr[\text{L-IND-CPA}^0_{A,N} = 1] - \Pr[\text{L-IND-CPA}^1_{A,N} = 1]|$, where L-IND-CPA is in figure 4.

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

AMAC: A AMAC scheme is defined by tuple $(\mathrm{M.mac}, \mathrm{M.vrf})$. 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 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 $t \leftarrow \mathrm{M.mac}(k, l, m)$ gives $\mathrm{M.vrf}(k, l, m, t) = \mathit{true}$. The user is only allowed one mac query and locks may not repeat between users. Verification queries are only allowed after the encryption. The security of the 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 5.

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 : $\quad k_j \xleftarrow{\$} \mathcal{K}$	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 5: 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 figure 15 of GKP.

3.2.2 ADEM'

A ADEM' scheme is defined by 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 $\perp$ to indicate rejection. The correctness requirement is that for every combination of k, l and m we have A.dec'$(k, l,$ A.enc'$(k, l, m)) = m$. The user is only allowed one encryption query and locks may not repeat between users. Decryption queries are only allowed after the encryption. The security of the ADEM' is defined as $\mathbf{Adv}^{\text{l-ind-cca}}_{\text{ADEM}',A,N} = |\Pr[\text{L-IND-CCA}^0_{A,N} = 1] - \Pr[\text{L-IND-CCA}^1_{A,N} = 1]|$, where L-IND-CCA is in 6.

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 : $\quad k_j \xleftarrow{\$} \mathcal{K}$	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 6: 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 figure 9 of GKP.

3.2.3 Construction

The ADEM' scheme considered is made by creating A.enc' and A.dec' calls using the calls the primitives provide us as seen in figure 7.

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

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

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.

4 Defining the new primitive

In this section we will discuss a new security primitive, the lock-based one time use Authenticated Encryption scheme, loAE scheme for short. As the name suggests, this primitive is used in a setting where a key is used only once to encrypt and authenticate a single message. We uses locks instead of nonces, as you will never have to decrypt messages with multiple nonces for a single user. Below, we discuss the notation of the loAE.

4.1 loAE

A loAE scheme is defined by tuple (AE.enc, AE.dec). Deterministic algorithm AE.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 AE.dec takes three inputs (k, n, c) and outputs a value m. Both AE.enc and EA.dec are required to satisfy correctness (if $\text{AE.enc}(k, l, m) = c \neq \perp$, then $\text{AE.dec}(k, l, c) = m$) and tidiness (if $\text{AE.dec}(k, l, c) = m \neq \perp$, then $\text{AE.enc}(k, l, m) = c$).

4.2 Security Model

To define the security, we use a ind\$ security notion instead of left-or-right one as, in our setting, ind\$ is the stronger security notion. (**todo: add citation**) We use a function that always returns $\perp$ on decryption calls to ensure the adversary can not guess which ciphertexts would be valid ciphertexts. To match our setting, the user is only allowed one encryption query, locks may not repeat between users and decryption queries are only allowed after the encryption.The security is defined as $\mathbf{Adv}^{\text{loAE}}_{A,N} = |\Pr[\text{loAE-IND\$}^0_{A,N} = 1] - \Pr[\text{loAE-IND\$}^1_{A,N} = 1]|$, where loAE-IND\$ is in figure 3.

Game $\text{loAE-IND\$}^b_{A,N}$

0 : $L \leftarrow \emptyset$
1 : $\textbf{for } j \in [1..N] :$
2 : $\quad k_j \xleftarrow{\$} \mathcal{K}$
3 : $\quad C_j \leftarrow \perp$
4 : $b' \leftarrow A$
5 : $\textbf{return } b'$

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

6 : $\textbf{if } C_j \neq \perp : \textbf{return } \perp$
7 : $\textbf{if } l \in L : \textbf{return } \perp$
8 : $L \leftarrow L \cup \{l\}$
9 : $l_j \leftarrow l$
10 : $c \leftarrow \text{AE.enc}(k_j, l_j, m)$
11 : $\textbf{if } b = 1 \wedge c \neq \perp :$
12 : $\quad c \xleftarrow{\$} \{0,1\}^{|c|}$
13 : $C_j \leftarrow c$
14 : $\textbf{return } c$

Oracle $\text{Odec}(j, c)$

15 : $\textbf{if } b = 1 : \textbf{return } \perp$
16 : $\textbf{if } C_j = \perp : \textbf{return } \perp$
17 : $\textbf{if } c = C_j : \textbf{return } \perp$
18 : $m \leftarrow \text{AE.dec}(k_j, l_j, c)$
19 : $\textbf{return } m$

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

4.3 Explanation of the Security Model

In this subsection we will discuss the security game loAE-IND\$ line by line.

Draft:
line 0: We start by initializing the set of all used locks to the empty set. *line 1:* Next we loop over each user to initialize them. *line 2:* For each user, we sample a uniformly random key from the key space and assign it to that user-key. *line 3:* for each user, we initialize the user-ciphertext to be undefined. We do not use set notation for this as done in GKP, as we can never have multiple ciphertexts related to one user. *line 4:* After initializing all users, we run the algorithm of the adversary and safe the output. *line 5:* We end by returning this output. *line 6:* We do not allow multiple encryptions per user. Therefore, if the ciphertexts is defined we return $\perp$. *line 7:* Locks are not allowed to repeat, if the lock is in the set of used sets we return $\perp$. *line 8:* If these checks pass, we start the encryption by adding the lock to the sets of used locks *line 9:* We bind the lock to the user-lock. *line 10:* We encrypt the message using the user-key and the user-lock. *line 11:* The random function should return $\perp$ whenever AE returns $\perp$. Therefore the random function is only called if b = 1 and AE does not return $\perp$. *line 12:* The random function samples a string with the uniformly at random from the set of all strings with the length of the ciphertext. **(todo: add part about ideal vs attainable)** *line 13:* The computed ciphertext is bound to the user-ciphertext. **(todo: add part about the fact that this might be $\perp$)** *line 14:* Return the ciphertext. *line 15:* The ideal world always returns $\perp$. *line 16:* If the user-ciphertext is not defined, decryption is not allowed and we return $\perp$. *line 17:* If the input ciphertext of Odec is allowed to be the same as the user-ciphertext, then it is trivial to distinguish as in this case the ideal words will return $\perp$ while the real world does not. For this reason the real world should return $\perp$ as well. *line 18:* If these checks pass, we decrypt the ciphertext. *line 19:* We return the decrypted message.

5 Constructions

In this section we discuss how we can construct a safe loAE. Similarly to GKP and NRS we will look at constructions combining a deterministic encryption primitive and mac primitive. First 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

loE: A lock-based one time use encryption scheme, loE for short, is defined by 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 and E.dec are required to satisfy correctness (if $\text{E.enc}(k, l, m) = c \neq \perp$, then $\text{E.dec}(k, l, c) = m$) and tidiness (if $\text{E.dec}(k, l, c) = m \neq \perp$, then $\text{E.enc}(k, l, m) = c$). The user is only allowed one encryption query and locks may not repeat between users. Decryption queries are only allowed after the encryption. The security is defined as $\mathbf{Adv}^{\text{loE}}_{A,N} = |\Pr[\text{loE-IND\$}^0_{A,N} = 1] - \Pr[\text{loE-IND\$}^1_{A,N} = 1]|$, where loE-IND$ is in figure 9.

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

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

loMAC: A lock-based one time use MAC is 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 $\perp$. If, and only if, (k, l, m) is not in $\mathcal{K} \times \mathcal{L} \times \mathcal{M}$, t will be $\perp$. The user is only allowed one mac query and locks may not repeat between users. Verification queries are only allowed after the encryption. 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 in figure 10.

Game $\text{loMAC-PRF}^b_{A,N}$	**Oracle** $\text{Omac}(j, l, m)$	**Oracle** $\text{Ovrf}(j, m, t)$
$L \leftarrow \emptyset$	**if** $T_j \neq \bot$: **return** $\bot$	**if** $T_j = \bot$: **return** $\bot$
for $j \in [1..N]$:	**if** $l \in L$: **return** $\bot$	**if** $(m, t) = T_j$: **return** $\bot$
$\quad k_j \xleftarrow{\$} \mathcal{K}$	$L \leftarrow L \cup \{l\}$	**if** $b = 1$: **return** *false*
$\quad T_j \leftarrow \bot$	$l_j \leftarrow l$	$t' \leftarrow \text{M.mac}(k_j, l_j, m)$
$b' \leftarrow A$	$t \leftarrow \text{M.mac}(k_j, l_j, m)$	**if** $t = t'$
return b'	**if** $b = 1 \land t \neq \bot$:	$\quad$ **return** *true*
	$\quad t \xleftarrow{\$} \{0,1\}^{\lvert t \rvert}$	**return** *false*
	$T_j \leftarrow (m, t)$	
	return t	

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

5.2 Construction

Following NRS, three ways to construct this loAE are of interest, namely the ones following from the N1, N2 and N3 scheme (N2 also corresponding to the construction from GKP). 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. The schemes, adjusted to our setting, are in figure 11. The AE.enc and AE.dec calls corresponding to N1, N2 and N3 are in figure 12, 13 and 14 respectively.

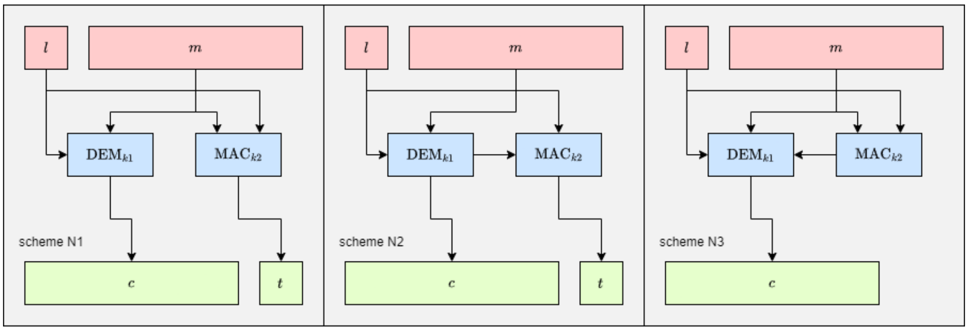

Figure 11: Adjusted N schemes from NRS

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

Figure 12: Calls based on N1

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

Figure 13: 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 14: Calls based on N3

5.3 Security Bounds

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

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 A

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

Name	GKP	NRS	my notation	rough meaning
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	-	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	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
adversary	A	$\mathcal{A}$	A	the bad guy
random sampling	$\xleftarrow{\$}$	$\twoheadleftarrow$	$\xleftarrow{\$}$	get a random ellermetn form the set
result of randomized function	$\xleftarrow{\$}$	-	$\leftarrow$	get the result of a randomized function with given inputs