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 whenever 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 $x \parallel y$, 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 write N for the number of users and allow a single type of error message written as $\perp$. We assume the existence of a nonempty key space $\mathcal{K}$, a lock space $\mathcal{L}$, a nonce space $\mathcal{N}$, a message space $\mathcal{M}$ and a associated-data space $\mathcal{A}$. Unless stated otherwise, $\mathcal{M}$ 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.

2.2 Authenticated Encryption

Two different security requirements are data privacy, the insurance that data cannot be viewed by a unauthorized party, and data integrity, the insurance that data has not been modified by a unauthorized party. Authenticated encryption combines both of these security requirements into one and ensures both data privacy and integrity. In addition some authenticated encryption schemes allow you to provide additional data, AD. This 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 Nonces and Locks improved

Both nonces and locks are given as input to an encryption scheme. They prevent a message from leading to the same ciphertext when encrypted multiple times under the same key. Although they look similar, their purpose and exact working differ leading to different use cases.

Nonces Normally, when a message is encrypted twice by the same user, it results in the same ciphertext. This can leak information about the message, which can be prevented 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 send 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 Normally, when two users that have a key collision encrypt the same message, results in the the same ciphertext. This can leak information about the secret keys used, which can be prevented using locks. Whereas nonces are bound to the message, locks are bound to the user. Each user has one lock and will encrypt all their messages using that lock. (**Question: Locks are not exclusive to a one-time use key right?**) 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 a adversary to see if two users have a key collision unless locks collide as well. To prevent this 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 one user.

2.4 Security Notions

The security of a cryptographic construction can be modeled as a distinguishing advantage. There are different things we can distinguish on, leading to different security notions. The relevant ones are written below.

IND-CPA vs IND-CCA You can model against an passive or an active attacker. A passive attacker can only read the messages while a active attacker can also alter the messages. A 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. A 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.

IND-$ vs IND-LOR 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 random function $. This random function returns a random string with the same length as the ciphertext would have. The adversary has to guess which of these two it has access to. As long as the length of the ciphertext is not randomized, IND-$ implies IND-LOR, but not the other way around. (**todo: add citation**)

Both of these are separate dimension and can be combined into 4 different notations. 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.

PRF-MAC vs unforgeable MAC A MAC function is said to be PRF secure when it is infeasible to distinguish the tag it outputs from the result of a PRF taking its input space to the

tag space. A MAC is said to be unforgeable when it is infeasible to create a valid message-tag pair without using 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.

2.5 Game Based Security Notions

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, you 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 be more comprehensible and precise. (**todo: add part about game hops**)

2.6 Security Proofs

general introduction to how we proof security of generic composition.

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 papers. 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 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 (**todo add citation for Automated Analysis of Protocols that use Authenticated Encryption: How Subtle AEAD**) in order to better match the notation from GKP and be more adaptable to a multi-user setting.

3.1 NRS

Three generic ways to construct an authenticated encryption scheme are discussed in a paper written by Bellare and Namprempre [2]: 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. We will look at the constructions using a nonce-based encryption scheme, nE for short, and a PRF secure MAC function.

3.1.1 Used Primitives

nE A nonce-based encryption scheme is defined by a triple $\Pi = (\mathcal{K}, \mathrm{E}, \mathrm{D})$. 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

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

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

Game MAC-PRF^b_A	**Oracle** Omac(m)
0 : $U \leftarrow \emptyset$	4 : **if** $m \in U$: **return** $\perp$
1 : $k \xleftarrow{\$} \mathcal{K}$	5 : $U \leftarrow U \cup \{m\}$
2 : $b' \leftarrow A$	6 : $t \leftarrow \mathrm{F}(k, m)$
3 : **return** b'	7 : **if** $b = 1 \wedge t \neq \perp$:
	8 : $t \xleftarrow{\$} \{0,1\}^{\lvert t \rvert}$
	9 : **return** t

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

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

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

where nE-IND-$-CPA is in figure 1. The adversary is not allowed to repeat nonces, set U keeps track of all used nonces.

MAC A MAC is defined by a 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 The security of a MAC 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 wins.

3.1.2 Nonce-Based Authenticated Encryption

A nonce-based authenticated encryption scheme is defined by a 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

Game nAE-IND-$-CCA$^b_A$

0 : $U \leftarrow \emptyset$
1 : $Q \leftarrow \emptyset$
2 : $k \xleftarrow{\$} \mathcal{K}$
3 : $b' \leftarrow A$
4 : **return** b'

Oracle Oenc(n, a, m)

6 : **if** $n \in U$: **return** $\perp$
7 : $U \leftarrow U \cup \{n\}$
8 : **if** $(n, a, m, \_) \in Q$: **return** $\perp$
9 : $c \leftarrow \mathrm{E}(k, n, a, m)$
10 : **if** $b = 1 \wedge c \neq \perp$:
11 : $\quad c \xleftarrow{\$} \{0,1\}^{|c|}$
12 : $Q \leftarrow Q \cup \{(n, a, m, c)\}$
13 : **return** c

Oracle Odec(n, a, c)

14 : **if** $b = 1$: **return** $\perp$
15 : **if** $(n, a, \_, c) \in Q$: **return** $\perp$
16 : $m \leftarrow \mathrm{D}(k, n, a, c)$
17 : $Q \leftarrow Q \cup \{(n, a, m, c)\}$
18 : **return** m

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

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

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

, where nAE-IND-$-CCA is in figure 3. The adversary is not allowed to repeat nonces on encryption, set U keeps track of all used nonces. Following the translation of IND-$-CCA to a security game for AE from (**todo add citation for Automated Analysis of Protocols that use Authenticated Encryption: How Subtle AEAD**), _ denotes a variable that is irrelevant and set Q keeps tack of all query results in order to prevent trivial wins.

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. This shows the notion from [2], stating 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 discussed. The authors start by showing some data encapsulation mechanisms are vulnerable to a passive multi-instance distinguishing- and key recovery and how this can lead to problems when used in public key encryption. They define the augmented data encapsulation mechanisms, ADEM for short, that uses locks to negate these insecurities. Additionally, they show 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.

Game L-IND-LOR-CPA$^b_{A,N}$	**Oracle** Oenc(j, l, m_0, m_1)
0 : $L \leftarrow \emptyset$	6 : **if** $C_j \neq \emptyset$: **return** $\perp$
1 : **for** $j \in [1..N]$:	7 : **if** $l \in L$: **return** $\perp$
2 : $k_j \xleftarrow{\$} \mathcal{K}$	8 : $L \leftarrow L \cup \{l\}$
3 : $C_j \leftarrow \emptyset$	9 : $l_j \leftarrow l$
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 4: L-IND-LOR-CPA game, A has access to oracle Oenc.

3.2.1 Used Primitives

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. 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.

ADEM A ADEM scheme is defined by tuple a (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$(k, l, m)) = m$.

ADEM security The security of a ADEM is defined as

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

(**Question: I was not sure to what extend should I explain the constructs from these games that I reuse and explain further in section 4.2, so I explained them briefly and wrote a note in the introduction of this section. What are your thoughts on this?**) 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 included a decryption oracle the adversary is not allowed to use.

AMAC A 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 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$(k, l, m, t) = true$.

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 5. 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

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 : $k_j \xleftarrow{\$} \mathcal{K}$	10 : $l_j \leftarrow l$	17 : forged $\leftarrow 1$
4 : $T_j \leftarrow \emptyset$	11 : $t \leftarrow$ M.mac(k_j, l_j, m)	18 : **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.

Game L-IND-LOR-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 : $k_j \xleftarrow{\$} \mathcal{K}$	8 : $L \leftarrow L \cup \{l\}$	15 : $m \leftarrow$ A.dec'(k_j, l_j, c)
3 : $C_j \leftarrow \emptyset$	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-LOR-CCA game, A has access to oracles Oenc and Odec and the locks in line 10 and 15 are the same.

queries are only allowed after the user made an mac query as enforced by line 4, 12 and 14. line 15 prevents trivial wins. The corresponding game can be found in figure 15 of GKP.

3.2.2 ADEM'

A 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 $\perp$ to indicate rejection. The correctness requirement is that for every combination of k, l and m we have A.dec'$(k, l,$ A.enc'$(k, l, m)) = m$.

ADEM' security The security of a ADEM' 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 6. 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 wins. The corresponding game can be found in figure 9 of GKP.

Proc A.enc'(k, l, m)	**Proc** A.dec'(k, l, c)
0 : $(k_{dem}, k_{mac}) \leftarrow k$	5 : $(k_{dem}, k_{mac}) \leftarrow k$
1 : $c' \leftarrow \text{A.enc}(k_{dem}, l, m)$	6 : $(c', t) \leftarrow c$
2 : $t \leftarrow \text{M.mac}(k_{mac}, l, c')$	7 : **if** $\text{M.vrf}(k_{mac}, l, c', t)$:
3 : $c \leftarrow (c', t)$	8 : $m \leftarrow \text{A.dec}(k_{dem}, l, c')$
4 : **return** c	9 : **return** m
	10 : **else** : **return** $\perp$

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

3.2.3 Construction

The ADEM' scheme considered is made by creating A.enc' and A.dec' using the tag-then-encrypt method from [2]. These new calls are in figure 7. 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 $\mathbf{Adv}^{\text{l-ind-lor-cca}}_{\text{ADEM'},A,N} \leq 2\mathbf{Adv}^{\text{l-miot-uf}}_{\text{AMAC},B,N} + \mathbf{Adv}^{\text{l-ind-lor-cpa}}_{\text{ADAM},C,N}$ 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.

3.3 Comparison of GKP and NRS

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

context and aim NRS is written in the context of symmetric cryptography. Because of this, a single-user that reuses its key is considered. GKP is written in the context of asymmetric cryptography. Because of this, multiple users that use their key once are considered. This difference in context leads to a different aim. While NRS is aimed at generalizing the generic AE constructions, GKP is aimed at finding a single construction that can be used for public key encryption. Most notably, this results in NRS evaluating 20 possible constructions while GKP evaluates one. Additionally, the constructions from NRS use AD while the construction from GKP does not.

Security Notion The security notions from both papers are also related to these different in contexts. In NRS, the security notions are written in a IND-$ fashion, common in symmetric cryptography. Conversely, in GKP, they are written in a IND-LOR fashion, common in asymmetric cryptography. 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 two papers have different security requirements. In NRS, the tag is required to be indistinguishable from a random string while in GKP the tag is required to be unforgeable. Furthermore, NRS considers nonces while GKP considers locks to match their settings.

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

0 : $L \leftarrow \emptyset$
1 : **for** $j \in [1..N]$:
2 : $\quad k_j \xleftarrow{\$} \mathcal{K}$
3 : $\quad C_j \leftarrow \perp$
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 : $c \leftarrow$ AE.enc(k_j, l_j, m)
11 : **if** $b = 1 \wedge c \neq \perp$:
12 : $\quad c \xleftarrow{\$} \{0,1\}^{|c|}$
13 : $C_j \leftarrow c$
14 : **return** c

Oracle Odec(j, c)

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

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

4 Defining the New Primitive

With the creation of a new primitive, we try to bridge the gap between NRS and GKP. Like GKP, we consider a authenticated encryption primitive in a setting where multiple users use their key once. But instead of finding a single constructions, we aim to generalize the primitive using the knowledge from NRS. In this section we will discuss this new security primitive, the lock-based one-time use Authenticated Encryption scheme, loAE scheme for short.

4.1 loAE

A loAE scheme is defined by tuple a (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 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$(k, l, m) = c$).

4.2 loAE Security Model

The security is defined as

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

where loAE-IND-$-CCA is in figure 8. Because we consider multiple user 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. We use locks as we want to account for key collisions between users, but not nonces as a user is only allowed to encrypt one message.

To define the security, we use IND-$ security notion instead of IND-LOR. It is the stronger security notion in our setting, as the length of the ciphertext is not randomized. On decryption, we use a function that always returns $\perp$ to ensure the adversary cannot guess which ciphertexts would be valid ciphertexts. 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 a invalid ciphertext in line 3. Whenever the adversary calls one of the oracles Oenc or Odec, it has to provide a user.

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 sets 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 ciphertexts 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 a 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 wins Line 16 prevents trivial wins. 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 samples as string uniformly at random from the set of all strings with the length of the ciphertext. This random string is bound it to the ciphertext in line 12. On decryption, the ideal world always returns $\perp$. **(todo: add part about ideal vs attainable)**

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, 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

loE A lock-based one-time use encryption scheme, loE for short, is defined by a tuple $(\text{E.enc}, \text{E.dec})$. 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

Game loE-IND-$-CPA$^b_{A,N}$	**Oracle** Oenc(j, l, m)
0 : $L \leftarrow \emptyset$	6 : **if** $C_j \neq \perp$: **return** $\perp$
1 : **for** $j \in [1..N]$:	7 : **if** $l \in L$: **return** $\perp$
2 : $k_j \xleftarrow{\$} \mathcal{K}$	8 : $L \leftarrow L \cup \{l\}$
3 : $C_j \leftarrow \perp$	9 : $l_j \leftarrow l$
4 : $b' \leftarrow A$	10 : $c \leftarrow \text{E.enc}(k_j, l_j, m)$
5 : **return** b'	11 : **if** $b = 1 \wedge c \neq \perp$:
	12 : $c \xleftarrow{\$} \{0,1\}^{\lvert c \rvert}$
	13 : $C_j \leftarrow c$
	14 : **return** c

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

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 $\text{E.enc}(k, l, m) = c$).

loE security The security of a loE is defined as

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

where loE-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.

loMAC A lock-based one-time use 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 $\perp$. If, and only if, (k, l, m) is not in $\mathcal{K} \times \mathcal{L} \times \mathcal{M}$, t will be $\perp$.

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

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

where loMAC-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 loMAC. This function takes the same input as the MAC would take, and outputs a uniformly random tag with the same length as the output tag from the MAC would have had.

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. The schemes, adjusted to our setting, are in figure 11. NRS

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

$L \leftarrow \emptyset$
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** $\perp$
$L \leftarrow L \cup \{l\}$
$l_j \leftarrow l$
$t \leftarrow \text{M.mac}(k_j, l_j, m)$
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** $\perp$
$t' \leftarrow \text{M.mac}(k_j, l_j, m)$
if $b = 1$:
  $t' \leftarrow tag(k_j, l_j, m)$
if $t = t'$
  return *true*
return *false*

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

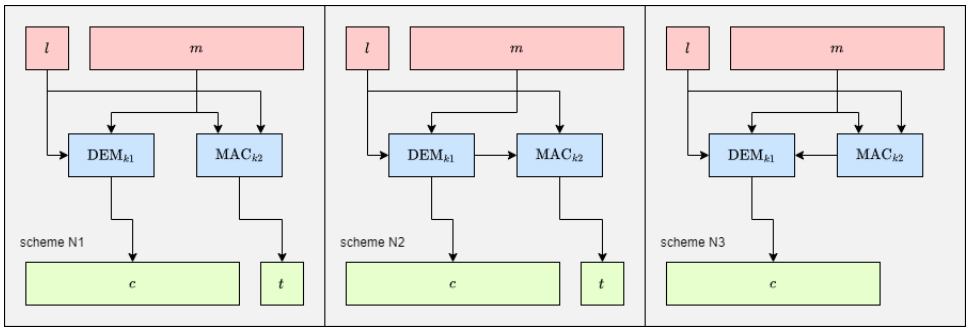

Figure 11: Adjusted N schemes from NRS

considers 17 more schemes but as no one 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 constructions secure if there is a tight (**todo: look at tight sec reductions**) reduction from breaking the loAE-security of the scheme to breaking the loE-security or the loMAC security of the underlying primitives. (**Note: This section is likely to contain some mistakes. I started trying to proof N2 as you said this would most likely be the easiest one. I plan to do N1 and N3 if you think the proof for N2 is correct.**)

AE.enc(k, l, m)	AE.dec(k, l, c)
0 : $(k1, k2) \leftarrow k$	5 : $(k1, k2) \leftarrow k$
1 : $c' \leftarrow \text{E.enc}(k1, l, m)$	6 : $(c', t) \leftarrow c$
2 : $t \leftarrow \text{M.mac}(k2, l, m)$	7 : $m \leftarrow \text{E.dec}(k1, l, c')$
3 : $c \leftarrow (c', t)$	8 : $t' \leftarrow \text{M.mac}(k2, l, m)$
4 : **return** c	9 : **if** $t \neq t' : m \leftarrow \perp$
	10 : **return** m

Figure 12: Calls based on N1

AE.enc(k, l, m)	AE.dec(k, l, c)
0 : $(k1, k2) \leftarrow k$	5 : $(k1, k2) \leftarrow k$
1 : $c' \leftarrow \text{E.enc}(k1, l, m)$	6 : $(c', t) \leftarrow c$
2 : $t \leftarrow \text{M.mac}(k2, l, c')$	7 : $m \leftarrow \text{E.dec}(k1, l, c')$
3 : $c \leftarrow (c', t)$	8 : $t' \leftarrow \text{M.mac}(k2, l, c')$
4 : **return** c	9 : **if** $t \neq t' : m \leftarrow \perp$
	10 : **return** m

Figure 13: Calls based on N2

AE.enc(k, l, m)	AE.dec(k, l, c)
0 : $(k1, k2) \leftarrow k$	5 : $(k1, k2) \leftarrow k$
1 : $t \leftarrow \text{M.mac}(k2, l, m)$	6 : $m' \leftarrow \text{E.dec}(k1, l, c)$
2 : $m' \leftarrow m \Vert t$	7 : $(m, t) \leftarrow m'$
3 : $c \leftarrow E.enc(k1, l, m')$	8 : $t' \leftarrow \text{M.mac}(k2, l, m)$
4 : **return** c	9 : **if** $t \neq t' : m \leftarrow \perp$
	10 : **return** m

Figure 14: Calls based on N3

N2 first, we define our theorem:

Theorem 1 *Let loAE be constructed from loMAC and loE as described in figure 13. Then for any number of users N and any loAE adversary A that poses at most Q_e many Oenc queries, limited by N, and at most Q_d many Odec queries, there exist a loMAC adversary B and a loE adversary C such that:*

$$\boldsymbol{Adv}^{loAE}_{A,N} = \boldsymbol{Adv}^{loMAC}_{B,N} + \frac{Q_d}{|t|^2} + \boldsymbol{Adv}^{loE}_{C,N} ,$$

where $|t|$ is the output length of the loMAC. The running time of B is at most that of A plus the time require 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. To prove this theorem we start by defining game loAE-N2 in figure 15. This game is the game loAE-IND-$-CCA (figure 8) AE.enc and AE.dec substituted with the N2 algorithms from figure 13.

Game loAE-N2$^b_{A,N}$	**Oracle** Oenc(j, l, m)	**Oracle** Odec(j, c)
0 : $L \leftarrow \emptyset$	6 : **if** $C_j \neq \perp$: **return** $\perp$	18 : **if** $C_j = \perp$: **return** $\perp$
1 : **for** $j \in [1..N]$:	7 : **if** $l \in L$: **return** $\perp$	19 : **if** $c = C_j$: **return** $\perp$
2 : $\quad k_j \xleftarrow{\$} \mathcal{K}$	8 : $L \leftarrow L \cup \{l\}$	20 : $(k1, k2) \leftarrow k_j$
3 : $\quad C_j \leftarrow \perp$	9 : $l_j \leftarrow l$	21 : $(c', t) \leftarrow c$
4 : $b' \leftarrow A$	10 : $(k1, k2) \leftarrow k_j$	22 : $m \leftarrow \text{E.dec}(k1, l_j, c')$
5 : **return** b'	11 : $c' \leftarrow \text{E.enc}(k1, l_j, m)$	23 : $t' \leftarrow \text{M.mac}(k2, l_j, c')$
	12 : $t \leftarrow \text{M.mac}(k2, l_j, c')$	24 : **if** $t \neq t'$: $m \leftarrow \perp$
	13 : $c \leftarrow (c', t)$	25 : **if** $b = 1$: $m \leftarrow \perp$
	14 : **if** $b = 1 \wedge c \neq \perp$:	26 : **return** m
	15 : $\quad c \xleftarrow{\$} \{0,1\}^{\lvert c \rvert}$	
	16 : $C_j \leftarrow c$	
	17 : **return** c	

Figure 15: loAE-N2 game, adversary has access to oracles Oenc and Odec.

This gives us

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

Next we define game N2-switch-1 in figure 16. The only difference between this game and game loAE-N2 is the fact that N2-switch-1 uses the uniformly random function *tag*, instead of the loMAC. This function takes the same input as the MAC would take, and outputs a uniformly random tag with the same length as the output tag from the MAC would have had.

Game N2-switch-$2^b_{A,N}$

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

Oracle Oenc(j, l, m)

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

Oracle Odec(j, c)

$18:$ **if** $C_j = \perp$: **return** $\perp$
$19:$ **if** $c = C_j$: **return** $\perp$
$20:$ $(k1, k2) \leftarrow k_j$
$21:$ $(c', t) \leftarrow c$
$22:$ $m \leftarrow \text{E.dec}(k1, l_j, c')$
$23:$ $t' \leftarrow tag(k2, l_j, c')$
$24:$ **if** $t \neq t'$: $m \leftarrow \perp$
$25:$ **if** $b = 1$: $m \leftarrow \perp$
$26:$ **return** m

Figure 16: N2-switch-1, adversary has access to oracles Oenc and Odec.

Using this game, we expand the probability:

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

Any adversary A winning game loAE-N2, but not N2-switch-1, can break the security of the loMAC. Concretely, we define adversary B against loMAC that wins security game loMAC-PRF (figure 10) using A.

Adverary B

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

if A calls **Oracle** Oenc(j, l, m)

$6:$ **if** $C_j \neq \perp$: **return** $\perp$
$7:$ **if** $l \in L$: **return** $\perp$
$8:$ $L \leftarrow L \cup \{l\}$
$9:$ $l_j \leftarrow l$
$10:$ $c' \leftarrow \text{E.enc}(k_j, l_j, m)$
$11:$ $t \leftarrow \text{Omac}(j, l_j, c')$
$12:$ $c \leftarrow (c', t)$
$13:$ $C_j \leftarrow c$
$14:$ **return** c

if A calls **Oracle** Odec(j, c)

$15:$ **if** $C_j = \perp$: **return** $\perp$
$16:$ **if** $c = C_j$: **return** $\perp$
$17:$ $(c', t) \leftarrow c$
$18:$ $m \leftarrow \text{E.dec}(k_j, l_j, c')$
$19:$ $passed \leftarrow \text{Ovrf}(j, c', t')$
$20:$ **if** $\neg passed$: $m \leftarrow \perp$
$21:$ **return** m

Figure 17: Adversary B, has access to A and oracles Omac and Ovrf. Note that k_j is the MAC key in this game.

This adversary has the runtime of A. For every Oenc query A makes, B computes E.enc once, and call 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. Using B, we can see that $\Pr[\text{loMAC-PRF}^0_{A,N} = 0] = \Pr[\text{loAE-N2}^0_{A,N} = 0]$ as B simulates game loAE-N2 with $b = 0$ whenever its own b is 0. In addition, we can see that $\Pr[\text{loMAC-PFR}^1_{A,N} = 0] = \Pr[\text{N2-switch-1}^0_{A,N} = 0]$ as B simulates game N2-switch-1 with $b = 0$ whenever its own b is 1. As a result we can rewrite our probability to:

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

To expand our probability again, we define game N2-switch-2 in figure 18. The only difference between this game and the first switch game is that game 2 will always return $\perp$ when the Odec query is called. We use this game to expand our probability twice:

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

Game N2-switch-$2^b_{A,N}$

0 : $L \leftarrow \emptyset$
1 : **for** $j \in [1..N]$:
2 : $\quad k_j \xleftarrow{\$} \mathcal{K}$
3 : $\quad C_j \leftarrow \perp$
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 : $(k1, k2) \leftarrow k_j$
11 : $c' \leftarrow \text{E.enc}(k1, l_j, m)$
12 : $t \leftarrow tag(k2, l_j, c')$
13 : $c \leftarrow (c', t)$
14 : **if** $b = 1 \wedge c \neq \perp$:
15 : $\quad c \xleftarrow{\$} \{0,1\}^{|c|}$
16 : $C_j \leftarrow c$
17 : **return** c

Oracle Odec(j, c)

18 : **return** $\perp$

Figure 18: N2-switch-2 game, adversary has access to oracles Oenc and Odec.

In the real world, the oracle Odec from N2-switch-1 returns $\perp$, unless the condition on line 24 is false. As a result, $\Pr[\text{N2-switch-1}^0_{A,N} = 0] - \Pr[\text{N2-switch-2}^0_{A,N} = 0]$ will be equal to the chance of this condition being true. As t and t' are the result of the uniformly random function tag, where the input of tag may not be equal when computing both, the chance of t and t' being equal will be $\frac{1}{|t|^2}$ for every Odec calls. Combining this with at most Q_d calls to the decryption

oracle, we get:

$$\begin{aligned}\mathbf{Adv}^{\mathrm{loAE}}_{A,N} \leq \mathbf{Adv}^{\mathrm{loMAC}}_{B,N} + \frac{Q_d}{|t|^2} + \Pr[\text{N2-switch-2}^0_{A,N} = 0] - \Pr[\text{N2-switch-2}^1_{A,N} = 0] \\ + \Pr[\text{N2-switch-2}^1_{A,N} = 0] - \Pr[\text{loAE-N2}^1_{A,N} = 0]\end{aligned}$$

In N2-switch-2, no information can ever be gained from making a Odec query. As a result any adversary A winning this game can break break the security of the loE as well. Concretely, we define adversary C against loE that wins game loE-IND-$-CPA using this A. Adversary C forwards all Oenc queries and disregards all Odec queries. The runtime of C is that of A and C makes Q_e many Oenc queries. This leads to:

$$\Pr[\text{N2-switch-2}^0_{C,N} = 0] - \Pr[\text{N2-switch-2}^1_{C,N} = 0] = \mathbf{Adv}^{\mathrm{loE}}_{C,N}$$

In addition to this, games N2-switch-2 and loAE-N2 are equal in the ideal world, resulting in $\Pr[\text{N2-switch-2}^1_{A,N} = 0] - \Pr[\text{loAE-N2}^1_{A,N} = 0] = 0$. Filling both these in gives us our final result:

$$\mathbf{Adv}^{\mathrm{loAE}}_{A,N} \leq \mathbf{Adv}^{\mathrm{loMAC}}_{B,N} + \frac{Q_d}{|t|^2} + \mathbf{Adv}^{\mathrm{loE}}_{C,N}$$

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.

Appendix A

(**todo: elaborate more on this table**)
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 ellement from the set
result of randomized function	$\xleftarrow{\$}$	-	$\leftarrow$	get the result of a randomized function with given inputs