ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Characterizing Notions For Secure Cryptographic Channels

Master Thesis

Nicolas Klose

March 18, 2021

Advisors: Prof. Dr. Kenny Paterson, Dr. Felix Günther

Applied Cryptography Group
Institute of Information Security
Department of Computer Science, ETH Zürich

Abstract

Symmetric encryption schemes, which allow encryption and decryption of messages using a shared secret key, are used today in a multitude of applications. Stateless schemes, which can provide privacy and integrity, are not sufficient for many scenarios. Adding state allows schemes to provide protection against replay, reorder and dropping attacks. In this thesis we extend some of the analysis on stateless schemes to the stateful scenario, introducing a security notion for channels which provides a stronger levels of privacy than existing notions. We also introduce a new security goal that we call progress-hiding, which ensures that an adversary cannot learn anything about the state of a channel which it was not observing for the entire duration of the communication. These ideas are extended to general correctness notions, allowing us to analyse schemes which use unreliable network protocols. Finally, we show that the record layers of TLS and DTLS 1.3 fulfil these new strong notions.

Contents

Chapter 1

Introduction

A central property of encryption schemes is privacy, which ensures that ciphertexts do not leak information about the messages they encrypt. There are many different formalizations of privacy, especially for stateless schemes. In Chapter 2 we will introduce some of these privacy notions for stateless schemes and analyse the relations between them.

Privacy is not the only requirement we have for encryption schemes. In many applications, we do not wish to send a single ciphertext, but instead a sequence of multiple ciphertexts. Encryption schemes which enable this are often referred to as channels. The sequence of ciphertexts and the possibility of decrypting multiple messages give us additional security requirement for channels. For example, we might wish to prevent replays, where a ciphertext is decrypted twice; reordering, where the order of decryption does not match the order of encryptions; or dropping, where ciphertexts are left out. All of this can be combined by requiring that the order of decryptions is exactly the same as the order of encryptions. Stateless schemes cannot prevent these sorts of attacks, instead we requite stateful schemes. Chapter 3 introduces privacy notion for stateful channels, including some new ones based on the stateless notions.

Another important feature of encryption schemes is integrity, which ensures that it is hard to forge ciphertexts. In Chapter 4 we will introduce integrity for stateless and stateful schemes, and show how to combine it with privacy. We will also define a new security notion \$-sfAE for channels which is stronger than the notions commonly used.

In order to motivate the use of \$-sfAE we will introduce progress-hiding in Chapter 5,. This new security notion ensures that the state of a channel is hidden from an adversary which is absent for some period of time. This notion is guaranteed by \$-sfAE but not by the notions commonly used for channels.

In Chapter 6 we will apply the notions defined so far to the real-world protocol TLS and show that the record layer of TLS 1.3 achieves \$-sfAE but TLS 1.2 does not.

On unreliable network protocols it is necessary to relax the strict correctness requirements of stateful encryption. Depending on the use-case there are a variety of desired behaviours. In order to extend our results to all of these, we introduce predicates in Chapter 7. This will allow us to define and relate notions for privacy, integrity and correctness abstractly for any correctness notion. These ideas are applied to progress-hiding in Chapter 8.

Finally, we will show in Chapter 9 that the record protocol of DTLS 1.3 achieves the notion corresponding to \$-sfAE for an appropriate predicate.

1.1 Notation

Let $s, t \in \{0,1\}^*$ be bitstrings. The length of s in bits is denoted $|s|$. The concatenation of strings s and t is denoted $s||t$ and their (bit-wise) XOR is denoted $s \oplus t$.

Integers (mostly used as counters) are always encoded as bitstings in binary, so we can apply the operation above as well as performing basic integer operations like addition.

For a distribution $\tilde{U}$ over a set U, let $u \leftarrow\!\$\, \tilde{U}$ represents sampling an element u randomly from $\tilde{U}$. We also write $u \leftarrow\!\$\, U$ to mean sampling an element uniformly at random from U.

We require multiple data structures. We assume that there exist unique encodings of these as bitstrings which allow the operations defined below to be executed in constant time.

We let $\varnothing$ denote an empty instance of each data structure. We will ensure that the further usage uniquely determines which data structure is meant.

Sets are unordered collection of elements and are denoted by curly brackets, so $\{s, t\}$ is the set containing s and t. The union $S \cup T$ of two sets S and T is defined to be the set which contains all of the elements contained in S or T. We will sometimes write $S \cup c$ to mean $S \cup \{c\}$ for a string c. The membership checks $c \in S$ returns true if c is contained in S and otherwise false.

Arrays are ordered collections and are denoted by round brackets, so (s, t) is the array containing s and t. Adding the element c to A (as the last element) is denoted by $A||c$. The length of an array, so the number of its elements, is denoted by $|A|$. Next to membership checks arrays also allow for look-ups. For an index $0 \leq i < |A|$, $A[i]$ returns the $(i+1)$-th element contained in A.

Look-up tables are sets where each element is a key-value pair of the form $\{c : m\}$ with the additional requirement that each key is unique. Unions are defined as above, but return an error if the union would add a repeated key. A look-up table E also (surprisingly) offers a look-up operation. For a pair $\{c : m\}$ contained in E the operation $E[c]$ returns the message m.

We assume that all of the operation defined above can be performed in constant time. This is of course a very non-trivial (as in false) assumption, but it greatly simplifies the analysis of runtimes, which is not the core topic of this thesis.

1.2 Symmetric encryption schemes

A stateless symmetric encryption scheme $\mathcal{SE} := (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}) := (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, l)$ for $\mathcal{K}, \mathcal{M}, \mathcal{C} \subset \{0,1\}^*$ consists of four algorithms. The algorithm KGen takes a security parameter k and returns a key $K \in \mathcal{K}$. The algorithm Enc takes a key K and a plaintext $m \in \mathcal{M}$ and returns a ciphertext $c \in \mathcal{C}$. The algorithm Dec takes a key and a ciphertext and returns a plaintext. We generally write $\mathsf{Enc}_K(m) := \mathsf{Enc}(K, m)$ and $\mathsf{Dec}_K(c) := \mathsf{Dec}(K, c)$. The length function l takes a the length of a plaintext and returns an integer. The algorithms KGen and Enc can be randomized, the other two are deterministic.

A stateful encryption scheme consists of the same algorithms, except that the encryption and decryption algorithm have an additional state input. This state is usually different for encryption and decryption and is updated independently. Each algorithm returns the updated state along with its usual output. The key generation algorithm outputs the initial states. Unless the state is important for an argument we generally assume that the state is implicitly handled correctly and omit it in security games. In particular, when we assign the return value of an algorithm, we actually mean assigning the return value without the state, so just the portion that a stateless algorithm would also return.

In [BDJR97] the authors use the term stateful encryption scheme to mean schemes with stateful encryption, but not stateful decryption. This is mostly equivalent to stateless schemes in terms of security. The notion of stateful encryption schemes used here was introduced in [BKN02] and relies on stateful decryption. This can actually provide more security than stateless schemes and is necessary to model channels, which ensure protection from replay, reorder and dropping of ciphertexts.

In order for schemes to be useful, we need them to be correct. Here we differentiate between correctness for stateless and stateful encryption schemes. For stateless schemes, we require that if we encrypt a plaintext and then decrypt the resulting ciphertext with the same key, then the plaintext output by the decryption algorithm is the same as the initial plaintext. For stateful schemes we relax this and only require a correct decryption if the ciphertexts are decrypted in exactly the order they were encrypted.

Definition 1.1
Let $\mathcal{SE}$ be an encryption scheme. We say $\mathcal{SE}$ fulfils stateless correctness if

$$\forall K \in \mathcal{K}, m \in \mathcal{M} : \mathsf{Dec}_K(\mathsf{Enc}_K(m)) = m.$$

Now let $m_1, \dots, m_n$ be a sequence of messages and let s_0 and s'_0 be the initial states of the encryption and decryption algorithms. For $i \in [1, n]$ calculate $(c_i, s_i) = \mathsf{Enc}_K(m_i, s_{i-1})$ and $(m'_i, s'_i) = \mathsf{Dec}_K(c_i, s'_{i-1})$. Then we say that $\mathcal{SE}$ fulfils stateful correctness if $\forall 1 \leq i \leq n : m_i = m'_i$.

Note that a stateful encryption scheme can fulfil the stateless correctness notion and that a scheme which fulfils stateless correctness automatically fulfils stateful correctness (we can apply the definition to a stateless algorithm by defining its state to be an empty vector). So this definition is to some degree independent of whether the scheme $\mathcal{SE}$ itself is stateful or stateless.

1.3 Security notions

Next to schemes being correct it is of course important to ensure that they are secure[citation needed]. Based on different kinds of real-world attacks we can define security notions, in which a theoretical adversary tries to succeed at such an attack in an idealized game. These idealized games often increase the capabilities of the adversary in comparison with real-world scenarios, yielding more secure notions.

Every notion has a "security goal", which represent the attacks we are trying to protect from and lead to different assumptions about the capabilities and goals of the adversaries. The privacy notions we define also have a "flavor", which represent different ways to check whether the adversary can break privacy. These notions therefore use the naming scheme FLAVOR-GOAL.

We use the code-based game-playing framework introduced in [BR06] to define security notions. A notion N is defined by a security game $\mathsf{N}_{\mathcal{SE}}(k, b)$, where $\mathcal{SE}$ is an encryption scheme, k is a security parameter and b is a bit. Every game begins with an initialization procedure $\mathsf{Init}()$. This query usually contains at least the generation of a key K. We assume that all variables defined in a procedure are global, so that K (and any other variables) can be used in the remaining procedures. After the initialization the adversary can interact with the encryption scheme using the other procedures of the game. At the end of the game, the adversary calls the finalization query Fin. This query returns a bit d, which is the output of the game. The adversary is trying to ensure that $b = d$.

Let $\mathsf{N}^{\mathcal{A}}_{\mathcal{SE}}(k,b)$ be the output of the game $\mathsf{N}_{\mathcal{SE}}(k,b)$ played by adversary $\mathcal{A}$. We define the advantage of $\mathcal{A}$ against $\mathcal{SE}$ as

$$\mathsf{Adv}^{\mathsf{N}}_{\mathcal{SE},\mathcal{A}}(k) = \Pr[\mathsf{N}^{\mathcal{A}}_{\mathcal{SE}}(k,1) = 1] - \Pr[\mathsf{N}^{\mathcal{A}}_{\mathcal{SE}}(k,0) = 1].$$

The advantage of an adversary depends greatly on its resources. For example an adversary with unlimited resources could brute-force many security games by trying every possible key or ciphertext. So we need to quantify the resources an adversary has at its disposal. Let $\mathfrak{A}^{\mathsf{N}}_{(t,q,\mu)}$ be the set of all adversaries for $\mathsf{N}_{\mathcal{SE}}(k,b)$ with running time at most t using q queries totalling μ bits.

The runtime t represents the runtime of the entire game, which includes the runtime of the adversary itself and of the queries made by the adversary. When a query requires two plaintexts as an input, we define the bit-length of the query to be the length of the longer plaintext. Finally, the number of queries q and their bitsize μ do not include the initialization and finalization query. These queries represent setting up the attack scenario and checking the result of the attack and are out of the control of the adversary. Excluding them also allows us to ignore some constant factors in relations.

We define the (k,t,q,μ)-security of SE as

$$\mathsf{Sec}^{\mathsf{N}}_{\mathcal{SE}}(k,t,q,\mu) = \max_{\mathcal{A}\in\mathfrak{A}^{\mathsf{N}}_{(t,q,\mu)}} \{\mathsf{Adv}^{\mathsf{N}}_{\mathcal{SE},\mathcal{A}}(k)\}.$$

1.4 Relations between notions

For an ordered pair of notion $\mathsf{N}_1, \mathsf{N}_2$ there four kinds of relations we can prove: reductions and separations, which can either be loose or tight.

Reductions are used to show that N_1 implies N_2. Formally this means that

$$\mathsf{Sec}^{\mathsf{N}_2}_{\mathcal{SE}}(k_2,t_2,q_2,\mu_2) \leq c \cdot \mathsf{Sec}^{\mathsf{N}_1}_{\mathcal{SE}}(k_1,t_1,q_1,\mu_1).$$

Usually c, k_1, t_1, q_1, μ_1 depend on k_2, t_2, q_2 and μ_2 in some way. If c is constant we call the reduction tight, if it depends on k_2, t_2, q_2 or μ_2 loose. To prove a reduction, we usually assume to have an adversary $\mathcal{A}_2 \in \mathfrak{A}^{\mathsf{N}_2}_{(t_2,q_2,\mu_2)}$ and construct an adversary $\mathcal{A}_1$ for the game ${\mathsf{N}_1}_{\mathcal{SE}}(k,b)$. This adversary runs $\mathcal{A}_2$, and react to its queries. This can involve queries in the N_1 game and other operations. We then show that $\mathcal{A}_1$ uses the resources claimed (so that $\mathcal{A}_1 \in \mathfrak{A}^{\mathsf{N}_1}_{(t_1,q_1,\mu_1)}$) and argue that

$$\mathsf{Adv}^{\mathsf{N}_2}_{\mathcal{SE},\mathcal{A}_2}(k) \leq c \cdot \mathsf{Adv}^{\mathsf{N}_1}_{\mathcal{SE},\mathcal{A}_1}(k).$$

As we started with an arbitrary adversary $\mathcal{A}_2$, the bound also holds for the maximum over all adversaries, which is exactly the definition of security.

In the last step, we often want to argue that $\mathcal{A}_1$ simulates the game $\mathsf{N2}_{\mathcal{SE}}(k, b')$ for $\mathcal{A}_2$. This means that the answers to $\mathcal{A}_2$'s queries are distributed the same way as they would be if it were playing the game $\mathsf{N2}_{\mathcal{SE}}(k, b')$. This then allows us to replace output probabilities of $\mathcal{A}_1$ with output probabilities of $\mathcal{A}_2$. A common case is where the value of b' is equal to b. In this case the advantage of the constructed adversary is equal to that of the adversary it is running.

Separations are used to show a notion N_1 does not imply another notion N_2. We do this by showing that there exists an encryption scheme $\mathcal{SE}'$ so that $\mathsf{Sec}^{\mathsf{N}_1}_{\mathcal{SE}'}(k_1, t_1, q_1, \mu_1)$ is small and $\mathsf{Sec}^{\mathsf{N}_2}_{\mathcal{SE}'}(k_2, t_2, q_2, \mu_2)$ is large. We usually start by assuming we have an encryption scheme $\mathcal{SE}$ for which $\mathsf{Sec}^{\mathsf{N}_1}_{\mathcal{SE}}(k_1, t_1, q_1, \mu_1)$ is small. We then construct a new scheme $\mathcal{SE}'$ based on $\mathcal{SE}$ so that

$$\mathsf{Sec}^{\mathsf{N}_1}_{\mathcal{SE}'}(k_1, t_1, q_1, \mu_1) \leq \mathsf{Sec}^{\mathsf{N}_1}_{\mathcal{SE}}(k_1, t_1, q_1, \mu_1).$$

Finally, we construct an adversary $\mathcal{A}_2 \in \mathfrak{A}^{\mathsf{N}_2}_{(t_2,q_2,\mu_2)}$ so that $\mathsf{Adv}^{\mathsf{N}_2}_{\mathcal{SE},\mathcal{A}_2}(k) \geq c$. If c is a (reasonably large) constant, the separation is tight. If it depends on k, t_2, q_2 or μ_2, it is loose. Note that this proof strategy depends on the assumption that there is a scheme which is secure for N_1. This assumption is actually not very strong, as we will see that all the notions defined in this thesis can be efficiently instantiated. Note that we could theoretically also assume that the scheme $\mathcal{SE}$ is secure for some other notion and use that notion to bound $\mathsf{Sec}^{\mathsf{N}_1}_{\mathcal{SE}'}(k_1, t_1, q_1, \mu_1)$.

Note that a tight relation implies a loose relation of the same type and that no relation of the other type holds. A loose relation only implies that the tight relation of the other type does not hold. So we are generally trying to prove one of three things:

- a tight reduction, which we write as $\mathsf{N}_1 \Rightarrow \mathsf{N}_2$
- a tight seperation, which we write as $\mathsf{N}_1 \not\Rightarrow \mathsf{N}_2$, or
- a loose reduction and a loose seperation, which we write as $\mathsf{N}_1 \rightarrow \mathsf{N}_2$.

If we consider both orderings for a pair of notions there four combinations we are interested in proving:

- They are equivalent, so $\mathsf{N}_1 \Rightarrow \mathsf{N}_2$ and $\mathsf{N}_2 \Rightarrow \mathsf{N}_1$
- One is loosely stronger, so $\mathsf{N}_1 \Rightarrow \mathsf{N}_2$, $\mathsf{N}_2 \rightarrow \mathsf{N}_1$
- One is strictly stronger, so $\mathsf{N}_1 \Rightarrow \mathsf{N}_2$ and $\mathsf{N}_2 \not\Rightarrow \mathsf{N}_1$
- They are incomparable, so $\mathsf{N}_1 \not\Rightarrow \mathsf{N}_2$ and $\mathsf{N}_2 \not\Rightarrow \mathsf{N}_1$

Chapter 2

Stateless chosen-plaintext and chosen-ciphertext security

2.1 Chosen-plaintext notions

The first security goal we consider is privacy under chosen-plaintext attacks (CPA), which ensures that ciphertexts do not leak any information about their corresponding plaintexts. This is supposed to prevent attacks from passive adversaries, which observe communication going over a network, but do not tamper with the messages. In real-world scenarios, attackers usually cannot choose which messages are encrypted, but they might know the content of some message or of parts of messages. So we allow the adversary to choose the messages which are encrypted. We define five different CPA notions, which we call flavors. These flavors differ in how an adversary can encrypt messages and how the adversary attempts to extract information from a ciphertext.

The adversary always has access to a challenge oracle, which somehow challenges the adversary to break the privacy of the scheme. The challenge oracle always returns a ciphertext, which we refer to as a challenge ciphertext. In some notions, the challenge oracle is the only way for the adversary to encrypt messages, so the oracle is called a combined encryption and challenge oracle. In other notions the adversary has access to a separate encryption oracle, which returns true encryption of single plaintexts. In these notions, the challenge query can only called once. We now present the origin and basic ideas of five flavors.

Semantic (SEM) security

Semantic security was introduced by Goldwasser and Micali in [GM84] for asymmetric encryption. We use the adaption for symmetric encryption introduced in [BDJR97]. The idea of semantic security is the following: consider the information an adversary can extract from a ciphertext about the encrypted plaintext. We say a scheme is semantically secure if this information is equal to the information the adversary could extract without the ciphertext. We model this as follows: the adversary has a true encryption oracle and a challenge oracle. In the single call of the challenge oracle, the adversary provides a distribution over the message space. The challenge procedure samples two messages and returns the encryption of the second one. In the finalisation query, the adversary provides a function f, which represents the information the adversary is trying to guess, and a guess for value of f for the b-th message. If the guess is correct and $b = 1$, the adversary has successfully extracted the information from the ciphertext. On the other hand, if $b = 0$, then the adversary was able to get the information without the ciphertext (as it only had encryption of the other message), so the scheme is secure and the adversary loses.

Definition 2.1 (SEM-CPA)
The semantic chosen-plaintext security of an encryption scheme $\mathcal{SE}$ is defined by $\mathsf{Sec}_{\mathcal{SE}}^{\mathsf{SEM\text{-}CPA}}(k, t, q, \mu)$ using the game $\mathsf{SEM\text{-}CPA}_{\mathcal{SE}}(k, b)$ depicted in Figure 2.1. The input for encryption queries is a plaintext $m \in \mathcal{M}$. The input for the challenge query $\tilde{\mathcal{M}}$ is a distribution over the message space with the restriction that all messages contained in the distribution must have the same length. The input to the finalization query consists of a function f which maps from the message space $\mathcal{M}$ to an arbitrary space $\mathcal{X}$ and an element $y \in \mathcal{X}$.

Find-then-guess (FtG) security

Find-then-guess security was introduced in [BDJR97] as an adaptation of polynomial security from [GM84] for symmetric encryption. The idea of find-then-guess security is that the adversary cannot tell which one of two ciphertexts were encrypted. Specifically, the adversary has access to a true encryption oracle and a challenge oracle. In the challenge query the adversary provides two plaintexts m_0 and m_1. The oracle returns the encryption of m_b. In the finalisation query, the adversary provides a bit which represents a guess on the value of b.

Definition 2.2 (FtG-CPA)
The find-then-guess chosen-plaintext security of an encryption scheme $\mathcal{SE}$ with security parameter k against adversaries with runtime t using q queries totalling μ bits is defined by $\mathsf{Sec}_{\mathcal{SE}}^{\mathsf{FtG\text{-}CPA}}(k, t, q, \mu)$ using the game $\mathsf{FtG\text{-}CPA}_{\mathcal{SE}}(k, b)$ depicted in Figure 2.1. The input for the encryption and challenge queries are plaintexts $m, m_0, m_1 \in \mathcal{M}$. The input to the finalization query is a bit d.

Left-or-right (LoR) security

Left-or-right security was introduced in [BDJR97]. It is similar to find-then-guess security, but the adversary is allowed multiple attempts to guess which one of two plaintexts is encrypted. The true encryption oracle is removed. Instead, the adversary has access to a combined encryption and challenge oracle, which function just like the challenge oracle in find-then-guess security but can be called multiple times. The finalisation query is again a guess for *b*.

Definition 2.3 (LoR-CPA)
The left-or-right chosen-plaintext security of an encryption scheme $\mathcal{SE}$ with security parameter k against adversaries with runtime t using q queries totalling μ bits is defined by $\mathsf{Sec}^{\mathsf{LoR\text{-}CPA}}_{\mathcal{SE}}(k,t,q,\mu)$ using the game $\mathsf{LoR\text{-}CPA}_{\mathcal{SE}}(k,b)$ depicted in Figure 2.1. The input for encryption queries are plaintexts $m_0, m_1 \in \mathcal{M}$. The input to the finalization query is a bit d.

Real-or-random-message (RoR) security

Real-or-random security was also introduced in [BDJR97]. Here, the adversary is trying to tell whether a provided message or a random message was encrypted. The adversary has access to a combined encryption and challenge oracle, which takes a message and either returns the encryption of the message or the encryption of a random message, depending on *b*. The finalisation query is again a guess for *b*.

Definition 2.4 (RoR-CPA)
The real-or-random-message chosen-plaintext security of an encryption scheme $\mathcal{SE}$ with security parameter k against adversaries with runtime t using q queries totalling μ bits is defined by $\mathsf{Sec}^{\mathsf{RoR\text{-}CPA}}_{\mathcal{SE}}(k,t,q,\mu)$ using the game $\mathsf{RoR\text{-}CPA}_{\mathcal{SE}}(k,b)$ depicted in Figure 2.1. The input for encryption queries are plaintexts $m \in \mathcal{M}$. The input to the finalization query is a bit d.

Real-or-random-ciphertext ($) security

Real-or-random-ciphertext security was introduced in [Rog02] as IND$. As we will define security goals using this flavor which provide integrity in addition to indistinguishability, we drop the IND to avoid confusion. The idea of this notion is similar to real-or-random-message security. Instead of sampling a random message when $b = 0$, a random ciphertext is sampled. The length of this random ciphertext is determined by the length function of the encryption scheme.

Definition 2.5 ($-CPA)
The real-or-random-ciphertext chosen-plaintext security of an encryption scheme $\mathcal{SE}$ with security parameter k against adversaries with runtime t using q queries

totalling μ bits is defined by $\mathsf{Sec}^{\text{\$-CPA}}_{\mathcal{SE}}(k,t,q,\mu)$ using the game $\text{\$-CPA}_{\mathcal{SE}}(k,b)$ depicted in Figure 2.1. The input for encryption queries are plaintexts $m \in \mathcal{M}$. The input to the finalization query is a bit d.

2.2 Relations between stateless chosen-plaintext notions

We now examine the relations between the CPA notions. The following results are adapted from [BDJR97] and its updated full version [BDJR00], except for the relations concerning \$-CPA, which are (implicitly) stated in [Rog02]. We begin by showing that the notions LoR-CPA and RoR-CPA are equivalent.

Theorem 2.6 (RoR-CPA ⇒ LoR-CPA, Theorem 1 from [BDJR00])
Let $\mathcal{SE}$ be a symmetric encryption scheme. There exists a constant c so that for any k, t, q and μ we have

$$\mathsf{Sec}^{\text{LoR-CPA}}_{\mathcal{SE}}(k,t,q,\mu) \leq 2 \cdot \mathsf{Sec}^{\text{RoR-CPA}}_{\mathcal{SE}}(k,t+c\mu,q,\mu)$$

Proof Assume $\mathcal{A}_{\text{LoR-CPA}} \in \mathfrak{A}^{\text{LoR-CPA}}_{(t,q,\mu)}$ is an adversary. We construct an adversary $\mathcal{A}_{\text{RoR-CPA}} \in \mathfrak{A}^{\text{RoR-CPA}}_{(t+c\mu,q,\mu)}$ for an appropriate c. The new adversary runs $\mathcal{A}_{\text{LoR-CPA}}$ and reacts to its queries with the following procedures:

```
A_RoR-CPA
1 : proc $\mathsf{Init}_{\text{LoR-CPA}}()$
2 :     $b' \leftarrow\$ \{0,1\}$
3 :     return $\mathsf{Init}_{\text{RoR-CPA}}()$

4 : proc $\mathsf{Enc}_{\text{LoR-CPA}}(m_0, m_1)$
5 :     $c \leftarrow \mathsf{Enc}_{\text{RoR-CPA}}(m_{b'})$
6 :     return $c$

7 : proc $\mathsf{Fin}_{\text{LoR-CPA}}(d)$
8 :     return $\mathsf{Fin}_{\text{RoR-CPA}}(d = b')$
```

We first show that $\mathcal{A}_{\text{RoR-CPA}}$ actually uses the resources claimed and then argue that $\mathsf{Adv}^{\text{RoR-CPA}}_{\mathcal{SE},\mathcal{A}_{\text{RoR-CPA}}}(k) = \frac{1}{2}\mathsf{Adv}^{\text{LoR-CPA}}_{\mathcal{SE},\mathcal{A}_{\text{RoR-CPA}}}(k)$. Note that $\mathcal{A}_{\text{RoR-CPA}}$ sends exactly one query for each query of $\mathcal{A}_{\text{LoR-CPA}}$ with the same bitsize, so the number of queries and their total bitsize are the same.

Now consider runtimes of $\mathcal{A}_{\text{LoR-CPA}}$ and $\mathcal{A}_{\text{RoR-CPA}}$. As both of them contain the runtime of $\mathcal{A}_{\text{LoR-CPA}}$ without its queries, the difference between their

Game $\mathsf{LoR\text{-}CPA}_{\mathcal{SE}}(k,b)$

```
proc $\mathsf{Init}_{\mathsf{LoR\text{-}CPA}}()$
  $K \leftarrow\$ \mathsf{KGen}(k)$
  return $k$

proc $\mathsf{Enc}_{\mathsf{LoR\text{-}CPA}}(m_0, m_1)$
  if $|m_0| \neq |m_1|$
    $c \leftarrow \perp$
  else
    $c \leftarrow \mathsf{Enc}_K(m_b)$
  return $c$

proc $\mathsf{Fin}_{\mathsf{LoR\text{-}CPA}}(d)$
  return $d$
```

Game $\mathsf{RoR\text{-}CPA}_{\mathcal{SE}}(k,b)$

```
proc $\mathsf{Init}_{\mathsf{RoR\text{-}CPA}}()$
  $K \leftarrow\$ \mathsf{KGen}(k)$
  return $k$

proc $\mathsf{Enc}_{\mathsf{RoR\text{-}CPA}}(m_1)$
  $m_0 \leftarrow\$ \{0,1\}^{|m_1|}$
  $c \leftarrow \mathsf{Enc}_K(m_b)$
  return $c$

proc $\mathsf{Fin}_{\mathsf{RoR\text{-}CPA}}(d)$
  return $d$
```

Game $\mathsf{\$\text{-}CPA}_{\mathcal{SE}}(k,b)$

```
proc $\mathsf{Init}_{\mathsf{\$\text{-}CPA}}()$
  $K \leftarrow\$ \mathsf{KGen}(k)$
  return $k$

proc $\mathsf{Enc}_{\mathsf{\$\text{-}CPA}}(m)$
  $c_0 \leftarrow\$ \{0,1\}^{l(|m|)}$
  $c_1 \leftarrow \mathsf{Enc}_K(m)$
  return $c_b$

proc $\mathsf{Fin}_{\mathsf{\$\text{-}CPA}}(d)$
  return $d$
```

Game $\mathsf{SEM\text{-}CPA}_{\mathcal{SE}}(k,b)$

```
proc $\mathsf{Init}_{\mathsf{SEM\text{-}CPA}}()$
  $K \leftarrow\$ \mathsf{KGen}(k)$
  $flag \leftarrow 0$
  return $k$

proc $\mathsf{Enc}_{\mathsf{SEM\text{-}CPA}}(m)$
  $c \leftarrow \mathsf{Enc}_K(m)$
  return $c$

proc $\mathsf{Chal}_{\mathsf{SEM\text{-}CPA}}(\tilde{\mathcal{M}})$
  if $flag = 1$
    $c \leftarrow \perp$
  else
    $flag \leftarrow 1$
    $m_0 \leftarrow\$ \tilde{\mathcal{M}}$
    $m_1 \leftarrow\$ \tilde{\mathcal{M}}$
    $c \leftarrow \mathsf{Enc}_K(m_1)$
  return $c$

proc $\mathsf{Fin}_{\mathsf{SEM\text{-}CPA}}(y, f)$
  return $y = f(m_b)$
```

Game $\mathsf{FtG\text{-}CPA}_{\mathcal{SE}}(k,b)$

```
proc $\mathsf{Init}_{\mathsf{FtG\text{-}CPA}}()$
  $K \leftarrow\$ \mathsf{KGen}(k)$
  $flag \leftarrow 0$
  return $k$

proc $\mathsf{Enc}_{\mathsf{FtG\text{-}CPA}}(m)$
  $c \leftarrow \mathsf{Enc}_K(m)$
  return $c$

proc $\mathsf{Chal}_{\mathsf{FtG\text{-}CPA}}(m_0, m_1)$
  if $flag = 1$ or $|m_0| \neq |m_1|$
    $c \leftarrow \perp$
  else
    $flag \leftarrow 1$
    $c \leftarrow \mathsf{Enc}_K(m_b)$
  return $c$

proc $\mathsf{Fin}_{\mathsf{FtG\text{-}CPA}}(d)$
  return $d$
```

Figure 2.1: The games for the chosen-plaintext notions

runtimes is equal to the difference between the runtime of $\mathcal{A}_{\textsf{RoR-CPA}}$ (including the runtime of its queries in the RoR-CPA game) and the runtime of the queries of $\mathcal{A}_{\textsf{LoR-CPA}}$ in the LoR-CPA game. In the game played by $\mathcal{A}_{\textsf{RoR-CPA}}$ we are missing an if-statement from the LoR-CPA game to check if two messages are the same length. Instead, we have to the initialise b' and generate a random bit for each bit queried in an encryption query. As this last factor dominates the other two, we can bound the runtime for RoR-CPA by $t + c\mu$ as desired, where c represent the time to sample a random bit.

In the game $\textsf{RoR-CPA}_{\mathcal{SE}}(k, 0)$, the result of each $\textsf{Enc}_{\textsf{LoR-CPA}}(m_0, m_1)$ is the encryption of a random ciphertext and therefore independent of b'. So we have

$$\Pr[\textsf{RoR-CPA}_{\mathcal{SE}}^{\mathcal{A}_{\textsf{RoR-CPA}}}(k, 0) = 1] = \frac{1}{2}.$$

On the other hand, in the game $\textsf{RoR-CPA}_{\mathcal{SE}}(k, 1)$ each encryption query of $\mathcal{A}_{\textsf{LoR-CPA}}$ is answered with the encryption of $m_{b'}$. So the LoR-CPA game is simulated with bit b'. If $\mathcal{A}_{\textsf{LoR-CPA}}$ successfully guesses b', then $\mathcal{A}_{\textsf{RoR-CPA}}$ outputs 1, so

$$\begin{aligned}
&\Pr[\textsf{RoR-CPA}_{\mathcal{SE}}^{\mathcal{A}_{\textsf{RoR-CPA}}}(k, 1) = 1] = \Pr[\textsf{LoR-CPA}_{\mathcal{SE}}^{\mathcal{A}_{\textsf{LoR-CPA}}}(k, b') = 1] \\
&= \frac{1}{2}\Pr[\textsf{LoR-CPA}_{\mathcal{SE}}^{\mathcal{A}_{\textsf{LoR-CPA}}}(k, 1) = 1] + \frac{1}{2}\Pr[\textsf{LoR-CPA}_{\mathcal{SE}}^{\mathcal{A}_{\textsf{LoR-CPA}}}(k, 0) = 0] \\
&= \frac{1}{2}\Pr[\textsf{LoR-CPA}_{\mathcal{SE}}^{\mathcal{A}_{\textsf{LoR-CPA}}}(k, 1) = 1] + \frac{1}{2} - \frac{1}{2}\Pr[\textsf{LoR-CPA}_{\mathcal{SE}}^{\mathcal{A}_{\textsf{LoR-CPA}}}(k, 0) = 1]
\end{aligned}$$

All together we have

$$\begin{aligned}
&\textsf{Adv}_{\mathcal{SE}, \mathcal{A}_{\textsf{RoR-CPA}}}^{\textsf{RoR-CPA}}(k) \\
&= \frac{1}{2}\Pr[\textsf{LoR-CPA}_{\mathcal{SE}}^{\mathcal{A}_{\textsf{LoR-CPA}}}(k, 1) = 1] - \frac{1}{2}\Pr[\textsf{LoR-CPA}_{\mathcal{SE}}^{\mathcal{A}_{\textsf{LoR-CPA}}}(k, 0) = 1] \\
&= \frac{1}{2}\textsf{Adv}_{\mathcal{SE}, \mathcal{A}_{\textsf{LoR-CPA}}}^{\textsf{LoR-CPA}}(k),
\end{aligned}$$

as desired. □

Theorem 2.7 (LoR-CPA ⇒ RoR-CPA, Theorem 2 from [BDJR00])
Let $\mathcal{SE}$ be a symmetric encryption scheme. There exists a constant c so that for any k, t, q and μ we have

$$\textsf{Sec}_{\mathcal{SE}}^{\textsf{RoR-CPA}}(k, t, q, \mu) \leq \textsf{Sec}_{\mathcal{SE}}^{\textsf{LoR-CPA}}(k, t + cq, q, \mu).$$

Proof Assume $\mathcal{A}_{\textsf{RoR-CPA}} \in \mathfrak{A}_{(t,q,\mu)}^{\textsf{RoR-CPA}}$ is an adversary. We construct an adversary $\mathcal{A}_{\textsf{LoR-CPA}} \in \mathfrak{A}_{(t+cq,q,\mu)}^{\textsf{LoR-CPA}}$ for an appropriate c. The new adversary runs $\mathcal{A}_{\textsf{RoR-CPA}}$ and reacts to its queries with the following procedures:

```
$\mathcal{A}_{\mathsf{LoR\text{-}CPA}}$
proc $\mathsf{Init}_{\mathsf{RoR\text{-}CPA}}()$
    return $\mathsf{Init}_{\mathsf{LoR\text{-}CPA}}()$
proc $\mathsf{Enc}_{\mathsf{RoR\text{-}CPA}}(m_1)$
    $m_0 \leftarrow\$ \{0,1\}^{|m_0|}$
    $c \leftarrow \mathsf{Enc}_{\mathsf{LoR\text{-}CPA}}(m_0, m_1)$
    return $c$
proc $\mathsf{Fin}_{\mathsf{RoR\text{-}CPA}}(d)$
    return $\mathsf{Fin}_{\mathsf{LoR\text{-}CPA}}(d)$
```

The number of queries and their total bitsize are clearly the same. The difference in runtimes is exactly the difference between the encryption queries. For both adversaries a second random message m_1 is sampled, either by the adversary or by the oracle, and one message is encrypted. The only difference is therefore the if-statement checking whether m_0 and m_1 are the same length in the LoR-CPA game. So we can define c to be the runtime of this if-statement. This gives us the claimed runtime. Otherwise we would get a bound which also depends on μ.

Note that $\mathcal{A}_{\mathsf{LoR\text{-}CPA}}$ simulates the game $\mathsf{RoR\text{-}CPA}_{\mathcal{SE}}(k, b)$ for the same value of b as in the actual game $\mathsf{LoR\text{-}CPA}_{\mathcal{SE}}(k, b)$. So the output probabilties are equal and we have $\mathsf{Adv}^{\mathsf{LoR\text{-}CPA}}_{\mathcal{SE},\mathcal{A}_{\mathsf{LoR\text{-}CPA}}}(k) = \mathsf{Adv}^{\mathsf{RoR\text{-}CPA}}_{\mathcal{SE},\mathcal{A}_{\mathsf{RoR\text{-}CPA}}}(k)$. □

We now show that LoR-CPA is loosely stronger than FtG-CPA.

Theorem 2.8 (LoR-CPA ⇒ FtG-CPA, Theorem 3 from [BDJR00])
Let $\mathcal{SE}$ be a symmetric encryption scheme. There exists a constant c so that for any k, t, q and μ we have

$$\mathsf{Sec}^{\mathsf{FtG\text{-}CPA}}_{\mathcal{SE}}(k, t, q, \mu) \le \mathsf{Sec}^{\mathsf{LoR\text{-}CPA}}_{\mathcal{SE}}(k, t + cq, q, \mu).$$

Proof Assume $\mathcal{A}_{\mathsf{FtG\text{-}CPA}} \in \mathfrak{A}^{\mathsf{FtG\text{-}CPA}}_{(t,q,\mu)}$ is an adversary. We construct an adversary $\mathcal{A}_{\mathsf{LoR\text{-}CPA}} \in \mathfrak{A}^{\mathsf{LoR\text{-}CPA}}_{(t+cq,q,\mu)}$ for an appropriate c. The new adversary runs $\mathcal{A}_{\mathsf{FtG\text{-}CPA}}$ and reacts to its queries with the following procedures:

```
$\mathcal{A}_{\text{LoR-CPA}}$

proc $\mathsf{Init}_{\text{FtG-CPA}}()$
  $flag \leftarrow 0$
  return $\mathsf{Init}_{\text{LoR-CPA}}()$

proc $\mathsf{Enc}_{\text{FtG-CPA}}(m)$
  $c \leftarrow \mathsf{Enc}_{\text{LoR-CPA}}(m, m)$
  return $c$

proc $\mathsf{Chal}_{\text{FtG-CPA}}(m_0, m_1)$
  if $flag = 1$
    $c \leftarrow \perp$
  else
    $flag \leftarrow 1$
    $c \leftarrow \mathsf{Enc}_{\text{LoR-CPA}}(m_0, m_1)$
  return $c$

proc $\mathsf{Fin}_{\text{FtG-CPA}}(d)$
  return $\mathsf{Fin}_{\text{LoR-CPA}}(d)$
```

The number of queries and their total bitsize are clearly the same. The only difference in runtimes is that each $\mathsf{Enc}_{\text{LoR-CPA}}$ query contains an if-statement to compare the lengths of the two plaintexts, which is missing in the $\mathsf{Enc}_{\text{FtG-CPA}}$ queires. So we can bound the runtime of $\mathcal{A}_{\text{LoR-CPA}}$ by $t + cq$ as desired, where c is the runtime of the if-statement.

Note that $\mathcal{A}_{\text{LoR-CPA}}$ simulates the game $\text{FtG-CPA}_{\mathcal{SE}}(k, b)$ with the same value b as in the actual $\text{LoR-CPA}_{\mathcal{SE}}(k, b)$ game. So the output probabilities are equal and we have $\mathsf{Adv}^{\text{LoR-CPA}}_{\mathcal{SE},\mathcal{A}_{\text{LoR-CPA}}}(k) = \mathsf{Adv}^{\text{FtG-CPA}}_{\mathcal{SE},\mathcal{A}_{\text{FtG-CPA}}}(k)$. □

Theorem 2.9 (FtG-CPA → LoR-CPA, Theorem 4 from [BDJR00])
Let $\mathcal{SE}$ be a symmetric encryption scheme. There exists a constant c so that for any k, t, q and μ we have

$$\mathsf{Sec}^{\text{LoR-CPA}}_{\mathcal{SE}}(k, t, q, \mu) \leq q \cdot \mathsf{Sec}^{\text{FtG-CPA}}_{\mathcal{SE}}(k, t + cq, q, \mu).$$

Proof Assume $\mathcal{A}_{\text{LoR-CPA}} \in \mathfrak{A}^{\text{LoR-CPA}}_{(t,q,\mu)}$ is an adversary. We construct an adversary $\mathcal{A}_{\text{FtG-CPA}} \in \mathfrak{A}^{\text{FtG-CPA}}_{(t+cq,q,\mu)}$ for an appropriate c. The new adversary runs $\mathcal{A}_{\text{LoR-CPA}}$ and is defined on the left-hand side of Figure 2.2.

The number of queries and their total bitsize are clearly the same. The runtime is increased by the two variables in the initialisation query and the if-statements in each encryption query. This is clearly bounded by a constant c for each query. Now consider the advantage of this adversary. We bound it using a hybrid argument. For $i \in [0, q]$ consider the game G_i defined on the right-hand side of Figure 2.2.

Note that given the choice of i by $\mathcal{A}_{\text{FtG-CPA}}$, $\text{FtG-CPA}^{\mathcal{A}_{\text{FtG-CPA}}}_{\mathcal{SE}}(k, b)$ is either equal to $\mathsf{G}_{i-1}{}^{\mathcal{A}_{\text{FtG-CPA}}}_{\mathcal{SE}}(k)$ or $\mathsf{G}_{i}{}^{\mathcal{A}_{\text{FtG-CPA}}}_{\mathcal{SE}}(k)$, depending on b. Also,

$$\Pr[\mathsf{G}_0{}^{\mathcal{A}_{\text{FtG-CPA}}}_{\mathcal{SE}}(k) = 1] = \Pr[\text{LoR-CPA}^{\mathcal{A}_{\text{LoR-CPA}}}_{\mathcal{SE}}(k, 0) = 1]$$

```
$\mathcal{A}_{\mathsf{FtG\text{-}CPA}}$

proc $\mathsf{Init}_{\mathsf{LoR\text{-}CPA}}()$
    $i \leftarrow\!\$\ \{1, \ldots, q\}$
    $j \leftarrow 1$
    return $\mathsf{Init}_{\mathsf{FtG\text{-}CPA}}()$

proc $\mathsf{Enc}_{\mathsf{LoR\text{-}CPA}}(m_0, m_1)$
    if $j < i$
        $c \leftarrow \mathsf{Enc}_{\mathsf{FtG\text{-}CPA}}(m_0)$
    elseif $j = i$
        $c \leftarrow \mathsf{Chal}_{\mathsf{FtG\text{-}CPA}}(m_0, m_1)$
    else
        $c \leftarrow \mathsf{Enc}_{\mathsf{FtG\text{-}CPA}}(m_1)$
    $j \leftarrow j + 1$
    return $c$

proc $\mathsf{Fin}_{\mathsf{LoR\text{-}CPA}}(d)$
    return $\mathsf{Fin}_{\mathsf{FtG\text{-}CPA}}(d)$
```

```
Game $G_i$

proc $\mathsf{Init}_i$
    $j \leftarrow 0$
    $K \leftarrow\!\$\ \mathsf{KGen}(k)$

proc $\mathsf{Enc}_i(m_0, m_1)$
    if $j \leq i$
        $c \leftarrow \mathsf{Enc}_K(m_0)$
    else
        $c \leftarrow \mathsf{Enc}_K(m_1)$
    $j \leftarrow j + 1$
    return $c$

proc $\mathsf{Fin}_i(d)$
    return $d$
```

Figure 2.2: The adversary and hybrid game used in the proof of Theorem 2.9.

and

$$\Pr[\mathsf{G_q}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{FtG\text{-}CPA}}}(k) = 1] = \Pr[\mathsf{LoR\text{-}CPA}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{LoR\text{-}CPA}}}(k, 1) = 1].$$

So summing over each possible choice of i gives us

$$\begin{aligned}\mathsf{Adv}^{\mathsf{FtG\text{-}CPA}}_{\mathcal{SE},\mathcal{A}_{\mathsf{FtG\text{-}CPA}}}(k) &= \frac{1}{q}\sum_{i=1}^{q}\Big(\Pr[\mathsf{G_{i-1}}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{FtG\text{-}CPA}}}(k) = 1] - \Pr[\mathsf{G_i}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{FtG\text{-}CPA}}}(k) = 1]\Big)\\ &= \frac{1}{q}\Big(\Pr[\mathsf{G_0}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{FtG\text{-}CPA}}}(k) = 1] - \Pr[\mathsf{G_q}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{FtG\text{-}CPA}}}(k) = 1]\Big)\\ &= \frac{1}{q}\mathsf{Adv}^{\mathsf{LoR\text{-}CPA}}_{\mathcal{SE},\mathcal{A}_{\mathsf{LoR\text{-}CPA}}}(k),\end{aligned}$$

as desired. □

Theorem 2.10 (FtG-CPA $\not\Rightarrow$ LoR-CPA, Theorem 5 from [BDJR00])
Let $\mathcal{SE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme and $q \in \mathbb{N}$. There exists an encryption scheme $\mathcal{SE}'$ (depending on q) and a constant c so that for every k, t and μ we have

$$\mathsf{Sec}^{\mathsf{FtG\text{-}CPA}}_{\mathcal{SE}'}(k, t, q, \mu) \leq \mathsf{Sec}^{\mathsf{FtG\text{-}CPA}}_{\mathcal{SE}}(k, t + cq, q, \mu) + \frac{1}{q}$$

$$\mathsf{Sec}^{\mathsf{LoR\text{-}CPA}}_{\mathcal{SE}'}(k, t, q, \mu) = 1$$

Proof The idea behind $\mathcal{SE}'$ is that for one randomly chosen encryption query it outputs the plaintext instead of the ciphertext. This is insecure in the LoR-CPA game but only very rarely in the FtG-CPA game, as the random query has to line up with the challenge query. The scheme $\mathcal{SE}'$ uses stateful encryption. The state consists of a counter i and a fixed value j. The scheme works as follows:

Scheme $\mathcal{SE}'$

```
proc $\mathsf{KGen}_{\mathcal{SE}'}(k)$
  $K \leftarrow\!\$\ \mathsf{KGen}_{\mathcal{SE}}(k)$
  $i \leftarrow 0$
  $j \leftarrow\!\$\ \{1, \dots, q\}$
  return $(K, (i, j))$

proc $\mathsf{Enc}_{\mathcal{SE}'}(K, m, (i, j))$
  $i \leftarrow i + 1$
  if $j = i$
    $c \leftarrow 1 \| m$
  else
    $c \leftarrow 0 \| \mathsf{Enc}_{\mathcal{SE}}(K, m)$
  return $(c, (i, j))$

proc $\mathsf{Dec}_{\mathcal{SE}'}(K, c)$
  $b \| c' \leftarrow c$
  if $b = 1$
    $m \leftarrow c'$
  else
    $m \leftarrow \mathsf{Dec}_{\mathcal{SE}}(K, c')$
  return $m$
```

Now consider an adversary for the FtG-CPA game against $\mathcal{SE}'$. We can construct an adversary for the FtG-CPA game against $\mathcal{SE}$ that chooses j and updates i itself. Every query can be forwarded except the j-th encryption or challenge query, which is returned as in $\mathcal{SE}'$. As long the j-th encryption query is not the challenge query, this simulates the game against $\mathcal{SE}'$ with the same value b as in the actual game against $\mathcal{SE}$, so it has the same advantage. If the j-th query is the challenge query, we choose one of the two messages at random to return. In this case the adversary can achieve advantage 1. This adversary has a constant increase in runtime for each query, so it uses the claimed resources. Also, we have

$$\begin{aligned}\mathsf{Sec}^{\mathsf{FtG\text{-}CPA}}_{\mathcal{SE}'}(k, t, q, \mu) &= \frac{q-1}{q}\mathsf{Sec}^{\mathsf{FtG\text{-}CPA}}_{\mathcal{SE}}(k, t, q, \mu) + \frac{1}{q} \\ &\le \mathsf{Sec}^{\mathsf{FtG\text{-}CPA}}_{\mathcal{SE}}(k, t, q, \mu) + \frac{1}{q}.\end{aligned}$$

For the LoR-CPA game against $\mathcal{SE}'$ consider an adversary that chooses two distinct messages and sends them in each of the q encryption queries. This adversary uses the claimed resources. The j-th query reveals which one was encrypted, allowing the adversary to win every game. □

Note that if there exists a scheme $\mathcal{SE}$ so that $\mathsf{Sec}^{\mathsf{FtG\text{-}CPA}}_{\mathcal{SE}}(k,t,q,\mu)$ is very small, the above theorem tell us that

$$q \cdot \mathsf{Sec}^{\mathsf{FtG\text{-}CPA}}_{\mathcal{SE}'}(k,t,q,\mu) \approx 1 = \mathsf{Sec}^{\mathsf{LoR\text{-}CPA}}_{\mathcal{SE}'}(k,t,q,\mu),$$

so the bound in Theorem 2.9 is tight.

Before moving on, we make small remark on the scheme $\mathcal{SE}'$. Note that its ciphertexts are vectors instead of bitstrings. As we defined ciphertext spaces to be a subset of $\{0,1\}^*$, we actually mean the encoding of a vector as a bitstring. This encoding is in general distinguishable from random bits. So if we were to define a scheme which returns vectors as ciphertexts and try to show that it achieves \$-CPA we would have to argue how the encoding works.

We now show that FtG-CPA and SEM-CPA are equivalent.

Theorem 2.11 (SEM-CPA ⇒ FtG-CPA, Theorem 6 from [BDJR00])
Let $\mathcal{SE}$ be a symmetric encryption scheme. There exists a constants c_1 and c_2 so that for any k, t, q and μ we have

$$\mathsf{Sec}^{\mathsf{FtG\text{-}CPA}}_{\mathcal{SE}}(k,t,q,\mu) \leq 2 \cdot \mathsf{Sec}^{\mathsf{SEM\text{-}CPA}}_{\mathcal{SE}}(k,t+c_1,q,2\mu+c_2).$$

Proof Assume $\mathcal{A}_{\mathsf{FtG\text{-}CPA}} \in \mathfrak{A}^{\mathsf{FtG\text{-}CPA}}_{(t,q,\mu)}$ is an adversary. We construct an adversary $\mathcal{A}_{\mathsf{SEM\text{-}CPA}} \in \mathfrak{A}^{\mathsf{SEM\text{-}CPA}}_{(t+c_1,q,2\mu+c_2)}$ for an appropriate c. The new adversary runs $\mathcal{A}_{\mathsf{FtG\text{-}CPA}}$ and reacts to its queries with the following procedures where $\mathsf{unif}\{m_0,m_1\}$ is the uniform distribution over $\{m_0,m_1\}$ and $\mathsf{id}_{\mathcal{M}}$ is the identity function.

```
$\mathcal{A}_{\mathsf{SEM\text{-}CPA}}$
1 :  proc $\mathsf{Init}_{\mathsf{FtG\text{-}CPA}}()$
2 :    return $\mathsf{Init}_{\mathsf{SEM\text{-}CPA}}()$

3 :  proc $\mathsf{Enc}_{\mathsf{FtG\text{-}CPA}}(m)$
4 :    $c \leftarrow \mathsf{Enc}_{\mathsf{SEM\text{-}CPA}}(m)$
5 :    return $c$

6 :  proc $\mathsf{Chal}_{\mathsf{FtG\text{-}CPA}}(m_0,m_1)$
7 :    $\tilde{\mathcal{M}} \leftarrow \mathsf{unif}\{m_0,m_1\}$
8 :    $c \leftarrow \mathsf{Chal}_{\mathsf{SEM\text{-}CPA}}(\tilde{\mathcal{M}})$
9 :    return $c$

10 : proc $\mathsf{Fin}_{\mathsf{FtG\text{-}CPA}}(d)$
11 :   return $\mathsf{Fin}_{\mathsf{SEM\text{-}CPA}}(\mathsf{id}_{\mathcal{M}}, m_d)$
```

The number of queries is the same. The bitsize of the challenge query is increased by sending the distribution of two messages instead of the messages. We assume this is a constant increase, as we only need to add the probability $\frac{1}{2}$ to each message. The finalisation query is increased by sending a message and the identity function instead of a single bit. As the challenge query contained the message, we can (very generously) bound this by doubling

μ. We also assume that the identity function can be bounded by a constant bitsize. The remaining queries have the same bitsize, so the claimed bitsize is correct. The runtime is only increased by the sampling of two messages in the $\mathsf{Chal}_{\mathsf{SEM\text{-}CPA}}$ query, so we can take c_1 to be that time.

The $\mathsf{Chal}_{\mathsf{SEM\text{-}CPA}}$ query samples two messages m'_0, m'_1 and returns encryption the of m'_1. Note that

$$\Pr[\mathsf{SEM\text{-}CPA}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{SEM\text{-}CPA}}}(k,b)=1]=\Pr[m_d=m'_b].$$

Now consider the value of b in the SEM-CPA game. For $b=1$, the game $\mathsf{FtG\text{-}CPA}_{\mathcal{SE}}(k,d)$ is simulated with d such that that $m_d=m'_1$. So considering the two possible values of m'_1 gives us

$$\begin{aligned}&\Pr[\mathsf{SEM\text{-}CPA}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{SEM\text{-}CPA}}}(k,1)=1]\\&=\frac{1}{2}\Pr[\mathsf{FtG\text{-}CPA}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{FtG\text{-}CPA}}}(k,1)=1]+\frac{1}{2}\Pr[\mathsf{FtG\text{-}CPA}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{FtG\text{-}CPA}}}(k,0)=0]\end{aligned}$$

For $b=0$ we need to consider two cases. If $m'_0=m'_1$, then $\Pr[m_d=m'_0]=\Pr[m_d=m'_1]$. On the other hand if $m'_0\neq m'_1$, then

$$\begin{aligned}\Pr[m_d=m'_0]&=\Pr[m_d\neq m'_1]\\&=1-\Pr[m_d=m'_1].\end{aligned}$$

As each of these scenarios happens with probability $\frac{1}{2}$, we have

$$\Pr[\mathsf{SEM\text{-}CPA}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{SEM\text{-}CPA}}}(k,0)=1]=\frac{1}{2}$$

Combining the results for each value of b gives the desired bound on the advantage of $\mathcal{A}_{\mathsf{SEM\text{-}CPA}}$. □

Theorem 2.12 (FtG-CPA ⇒ SEM-CPA, Theorem 7 from [BDJR00])
Let $\mathcal{SE}$ be a symmetric encryption scheme. There exists a constant c so that for any k, t, q and μ we have

$$\mathsf{Sec}_{\mathcal{SE}}^{\mathsf{SEM\text{-}CPA}}(k,t,q,\mu)\leq 2\cdot\mathsf{Sec}_{\mathcal{SE}}^{\mathsf{FtG\text{-}CPA}}(k,t+cq,q,\mu).$$

Proof Assume $\mathcal{A}_{\mathsf{SEM\text{-}CPA}}\in\mathfrak{A}_{(t,q,\mu)}^{\mathsf{SEM\text{-}CPA}}$ is an adversary. We construct an adversary $\mathcal{A}_{\mathsf{FtG\text{-}CPA}}\in\mathfrak{A}_{(t+c,q,\mu)}^{\mathsf{FtG\text{-}CPA}}$ for an appropriate c. The new adversary runs $\mathcal{A}_{\mathsf{SEM\text{-}CPA}}$ and reacts to its queries with the following procedures:

```
$\mathcal{A}_{\mathsf{FtG\text{-}CPA}}$
proc $\mathsf{Init}_{\mathsf{SEM\text{-}CPA}}()$
    return $\mathsf{Init}_{\mathsf{FtG\text{-}CPA}}()$

proc $\mathsf{Enc}_{\mathsf{SEM\text{-}CPA}}(m)$
    $c \leftarrow \mathsf{Enc}_{\mathsf{FtG\text{-}CPA}}(m)$
    return $c$

proc $\mathsf{Chal}_{\mathsf{SEM\text{-}CPA}}(\tilde{\mathcal{M}})$
    $m_0 \leftarrow\$ \tilde{\mathcal{M}}$
    $m_1 \leftarrow\$ \tilde{\mathcal{M}}$
    $c \leftarrow \mathsf{Chal}_{\mathsf{FtG\text{-}CPA}}(m_0, m_1)$
    return $c$

proc $\mathsf{Fin}_{\mathsf{SEM\text{-}CPA}}(f, y)$
    if $f(m_1) = y$
        $d = 1$
    else
        $d \leftarrow\$ \{0,1\}$
    return $\mathsf{Fin}_{\mathsf{FtG\text{-}CPA}}(d)$
```

The number of queries and their bitsize are the same. The runtime is increased by the length comparison in $\mathsf{Chal}_{\mathsf{FtG\text{-}CPA}}$ and possibly by the random sampling of d, which we can clearly bound by a constant c. Note that the value (y, f) returned by $\mathsf{Fin}_{\mathsf{SEM\text{-}CPA}}$ does not depend on b

Now consider that value of b in the FtG-CPA game. For $b = 1$, $\mathcal{A}_{\mathsf{FtG\text{-}CPA}}$ simulates the game $\mathsf{SEM\text{-}CPA}_{\mathcal{SE}}(k, 1)$. So we have

$$\begin{aligned}
&\Pr[\mathsf{FtG\text{-}CPA}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{SEM\text{-}CPA}}}(k,1) = 1] \\
&= \Pr[\mathsf{SEM\text{-}CPA}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{SEM\text{-}CPA}}}(k,1) = 1] + \frac{1}{2}(1 - \Pr[\mathsf{SEM\text{-}CPA}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{SEM\text{-}CPA}}}(k,1) = 1]) \\
&= \frac{1}{2}(\Pr[\mathsf{SEM\text{-}CPA}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{SEM\text{-}CPA}}}(k,1) = 1] + 1)
\end{aligned}$$

For $b = 0$, $\mathcal{A}_{\mathsf{FtG\text{-}CPA}}$ simulates the game $\mathsf{SEM\text{-}CPA}_{\mathcal{SE}}(k, 1)$ except that in the query $\mathsf{Chal}_{\mathsf{SEM\text{-}CPA}}$, m_0 and m_1 are switched. So the encryption of m_0 is returned and the game outputs 1 if $y = f(m_1)$. This switch makes no difference for the probabilities, as both messages are sampled from the same distribution. So we have

$$\begin{aligned}
&\Pr[\mathsf{FtG\text{-}CPA}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{SEM\text{-}CPA}}}(k,0) = 1] \\
&= \Pr[\mathsf{SEM\text{-}CPA}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{SEM\text{-}CPA}}}(k,0) = 1] + \frac{1}{2}(1 - \Pr[\mathsf{SEM\text{-}CPA}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{SEM\text{-}CPA}}}(k,0) = 1]) \\
&= \frac{1}{2}(\Pr[\mathsf{SEM\text{-}CPA}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{SEM\text{-}CPA}}}(k,0) = 1] + 1)
\end{aligned}$$

Combining the results for each value of b gives the stated advantage. □

Finally, we show that \$-CPA is a strictly stronger notion than LoR-CPA.

Theorem 2.13 (\$-CPA $\Rightarrow$ LoR-CPA)
Let $\mathcal{SE}$ be a symmetric encryption scheme such that $l(|m|) \leq a|m|$ for a constant a. There exists a constant c so that for any k, t, q and μ we have

$$\mathsf{Sec}_{\mathcal{SE}}^{\mathsf{LoR\text{-}CPA}}(k,t,q,\mu) \leq 2 \cdot \mathsf{Sec}_{\mathcal{SE}}^{\mathsf{\$\text{-}CPA}}(k,t+c\mu,q,\mu).$$

Proof Assume $\mathcal{A}_{\mathsf{LoR\text{-}CPA}} \in \mathfrak{A}_{(t,q,\mu)}^{\mathsf{LoR\text{-}CPA}}$ is an adversary. We construct an adversary $\mathcal{A}_{\mathsf{\$\text{-}CPA}} \in \mathfrak{A}_{(t+cq,q,\mu)}^{\mathsf{\$\text{-}CPA}}$ for an appropriate c. The new adversary runs $\mathcal{A}_{\mathsf{LoR\text{-}CPA}}$ and reacts to its queries with the following procedures:

```
$\mathcal{A}_{\mathsf{\$\text{-}CPA}}$

1 : proc $\mathsf{Init}_{\mathsf{LoR\text{-}CPA}}()$
2 :   $b' \leftarrow\$ \{0,1\}$
3 :   return $\mathsf{Init}_{\mathsf{\$\text{-}CPA}}()$

4 : proc $\mathsf{Enc}_{\mathsf{LoR\text{-}CPA}}(m_0, m_1)$
5 :   $c \leftarrow \mathsf{Enc}_{\mathsf{\$\text{-}CPA}}(m_{b'})$
6 :   return $c$

7 : proc $\mathsf{Fin}_{\mathsf{LoR\text{-}CPA}}(d)$
8 :   return $\mathsf{Fin}_{\mathsf{\$\text{-}CPA}}(d = b')$
```

The number of queries and their bitsize are the same. The difference in runtime is dominated by the sampling of a random string in the \$-CPA game. By the requirement on the length function, we have to sample at most $a\mu$ bits, so we can bound this time by $c\mu$ for a c which also includes the time to sample a bit.

The argument to prove the stated advantage is identical to the argument in Theorem 2.6, so we omit it. $\square$

Theorem 2.14 (LoR-CPA $\not\Rightarrow$ \$-CPA)
Let $\mathcal{SE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme and take $n \in \mathbb{N}$. Then there exists an encryption scheme $\mathcal{SE}'$ and a constant c so that for any k, t, q and μ we have

$$\mathsf{Sec}_{\mathcal{SE}'}^{\mathsf{LoR\text{-}CPA}}(k,t,q,\mu) = \mathsf{Sec}_{\mathcal{SE}}^{\mathsf{LoR\text{-}CPA}}(k,t+cq,q,\mu)$$
$$\mathsf{Sec}_{\mathcal{SE}'}^{\mathsf{\$\text{-}CPA}}(k,t,1,\mu) = 1 - \frac{1}{2^{n+1}}$$

Proof Let 0^n be the string consisting of n 0-bits. We define the encryption algorithm Enc' of $\mathcal{SE}'$ by $\mathsf{Enc}'_K(m) = 0^n \parallel \mathsf{Enc}_K(m)$. The key generation algorithm is the same and the decryption algorithm works by first removing the prefix and then decrypting normally. We define the length function of $\mathcal{SE}'$ by $l'(|m|) = l(|m|) + n$.

Assuming we have adversary in the LoR-CPA game against $\mathcal{SE}'$, we can easily construct an adversary against $\mathcal{SE}$ that forwards all queries after adding or removing 0^n to ciphertexts appropriately. This adversary clearly has the same advantage as the original one and uses the claimed resources, so the first equality holds. For the second, consider an adversary in the \$-CPA game which sends a single query with an arbitrary message. If the returned string starts with 0^n, then it returns 0, otherwise 1. This adversary uses the claimed resources and only loses if the bit b in the \$-CPA game is 1 and the first n bits chosen at random are all 0. This happens with probability $\frac{1}{2^{n+1}}$.$\square$

2.3 Properties of chosen-plaintext-secure schemes

We now state some fairly basic attributes that a scheme must fulfil in order to be CPA secure for different flavors.

A scheme which is not randomized is not secure for any flavor. By the relations above, it suffices to show this for FtG-CPA security. So consider an adversary for the FtG-CPA game which first sends two encryption queries for two distinct messages, and then sends its challenge query with the two messages. If the scheme is not randomized, the challenge ciphertext is equal to one of the previous ciphertexts, allowing the adversary to learn the value of b. Indeed the probability that two encryptions of the same message are the same is a lower bound on the security of a scheme. This means not only do schemes need to be randomized, the amount of randomness needs to be large enough. Real-world schemes often use nonces instead of randomization. Nonce-based schemes are deterministic, but encryption and decryption receive an additional input which ensures that the encrypted ciphertexts are distinct as long as nonces are not repeated. We will formalize this in Chapter 6 for our analysis of TLS.

A scheme for a message space which is not closed under message length may not be secure for RoR-CPA notion (and therefore also for the LoR-CPA and \$-CPA). By closed under message length we mean that if the message space contains a string of specific bit length, then it contains every message of this length. In the encryption query of the RoR-CPA game, a random string is sampled with the length of the provided message. If the message space is not closed under message length, then it is possible for this string to be outside of the message space, resulting in undefined behaviour from the encryption scheme, which may leak information about b.

If the length of a ciphertext does not only depend on the length of the message, then the adversary might be able to learn something about the plaintext based on the length of its ciphertext, making it insecure for every flavor. In particular, the length function of the scheme should return the length of the ciphertext for messages of the input length, otherwise the scheme is insecure for $-CPA. Note that every flavor allows the ciphertext to leak the length of its plaintext without breaking the game. So these notions do not ensure that a scheme hides the length of plaintexts.

The next attribute is useful for proving separations, so we state it as a lemma.

Lemma 2.15
Let $\mathcal{SE}$ be an encryption scheme and let $\mathcal{A}$ be an adversary for a notion N which gives access to an $\mathsf{Chal}_{\mathsf{FtG\text{-}CPA}}$ oracle. Consider a ciphertext c which was returned from such a query. If the adversary can somehow gain knowledge of $\mathsf{Dec}_K(c)$, then $\mathsf{Sec}^{\mathsf{N}}_{\mathcal{SE}} = 1$.

Proof By sending a challenge query with two distinct messages and comparing them with $\mathsf{Dec}_K(c)$, the adversary can learn the value of b immediately. □

When we consider other security goals we will see that the relations between the flavors are the same. This lemma therefore allows us show that a scheme is insecure for all of the flavors at the same time by showing that an adversary with access to an $\mathsf{Chal}_{\mathsf{FtG\text{-}CPA}}$ oracle can learn the decryption of a challenge ciphertext. Unsurprisingly, every flavor is trivially insecure for adversary which can learn the decryption of challenge ciphertexts.

2.4 Chosen-ciphertext notions

The notions so far provided security against passive adversaries. We now wish to extend this to active adversaries, which can tamper with the ciphertexts. In real-world scenarios, the adversary usually cannot see the output of the decryption algorithm. But analysing responses and timing can leak some information about the decryption. So we would like to give the adversary access to a decryption oracle. As we just saw in Lemma 2.15, we cannot define meaningful games which give the adversary the capability to decrypt challenge ciphertexts. We can however give it the ability to decrypt any other ciphertext. This gives us a notion called security under chosen-ciphertext attacks (CCA). The CCA notions for the flavors LoR,RoR, FtG and SEM were first defined in [BDJR00], for $ in [Rog04].

For $\mathsf{FL} \in \{\mathsf{SEM}, \mathsf{FtG}, \mathsf{LoR}, \mathsf{RoR}, \$\}$ we define a CCA game where the adversary has access to the oracles of the CPA games and an additional decryption oracle, which takes a string as an input and outputs its decryption under the key generated in the initialization query. Note that we do not require that the adversary sends an actual ciphertext, unlike in the encryption queries, where we require that the adversary sends a valid plaintext. In order to prevent the decryption of challenge ciphertexts we maintain a set S of the challenge ciphertexts and reject decryption queries for ciphertexts contained in S.

Game $\mathsf{FL\text{-}CCA}_{\mathcal{SE}}(k, b)$

```
proc $\mathsf{Init}_{\mathsf{FL\text{-}CCA}}()$
    $S \leftarrow \varnothing$

    return $\mathsf{Init}_{\mathsf{FL\text{-}CPA}}()$

proc $\mathsf{Enc}_{\mathsf{FL\text{-}CCA}}(\overline{m})$
    $c \leftarrow \mathsf{Enc}_{\mathsf{FL\text{-}CPA}}(\overline{m})$

    $S \leftarrow S \cup \{c\}$
    return $c$

proc $\mathsf{Chal}_{\mathsf{FL\text{-}CCA}}(\overline{m})$
    $c \leftarrow \mathsf{Chal}_{\mathsf{FL\text{-}CPA}}(\overline{m})$
    $S \leftarrow S \cup \{c\}$
    return $c$

proc $\mathsf{Dec}_{\mathsf{FL\text{-}CCA}}(c)$
    if $c \in S$
        $m \leftarrow \perp$
    else
        $m \leftarrow \mathsf{Dec}_K(c)$
    return $m$

proc $\mathsf{Fin}_{\mathsf{FL\text{-}CCA}}(\overline{d})$
    return $\mathsf{Fin}_{\mathsf{FL\text{-}CPA}}(\overline{d})$
```

Game $\$\text{-}\mathsf{CCA}_{\mathcal{SE}}(k, b)$

```
proc $\mathsf{Init}_{\$\text{-}\mathsf{CCA}}()$
    $S \leftarrow \varnothing$
    $K \leftarrow\!\$\ \mathsf{KGen}(k)$
    return $k$

proc $\mathsf{Enc}_{\$\text{-}\mathsf{CCA}}(\overline{m})$
    $c_0 \leftarrow\!\$\ \{0,1\}^{l(|m|)}$
    $c_1 \leftarrow \mathsf{Enc}_K(m)$
    $S \leftarrow S \cup \{c_b\}$
    return $c_b$

proc $\mathsf{Dec}_{\$\text{-}\mathsf{CCA}}(c)$
    if $c \in S$
        $m \leftarrow \perp$
    else
        $m \leftarrow \mathsf{Dec}_K(c)$
    return $m$

proc $\mathsf{Fin}_{\$\text{-}\mathsf{CCA}}(\overline{d})$
    return $d$
```

Figure 2.3: The generic chosen-ciphertext game for $\mathsf{FL} \in \{\mathsf{SEM}, \mathsf{FtG}, \mathsf{LoR}, \mathsf{RoR}, \$\}$ and the game for \$-CCA.

Definition 2.16 (FL-CCA)
For $\mathsf{FL} \in \{\mathsf{SEM}, \mathsf{FtG}, \mathsf{LoR}, \mathsf{RoR}, \$\}$ *the chosen-ciphertext security of an encryption scheme* $\mathcal{SE}$ *with security parameter k against adversaries with runtime t using q queries totalling* μ *bits is defined by* $\mathsf{Sec}^{\mathsf{FL\text{-}CCA}}_{\mathcal{SE}}(k, t, q, \mu)$ *using the game* $\mathsf{FL\text{-}CCA}_{\mathcal{SE}}(k, b)$ *depicted in Figure 2.3. The input for encryption, challenge and finalisation queries are the same as in the corresponding* FL-CPA *game. The input to the decryption query is an arbitrary string c. The challenge queries are only defined if they exist in the corresponding* FL-CPA *game.*

The next theorem shows that the relations between the CPA notions also hold for the corresponding CCA notions. These results (excluding the flavor \$) are also shown in [BDJR00].

Theorem 2.17
The Theorems 2.6, 2.7, 2.8, 2.9, 2.10 and 2.11, 2.12, 2.13 and 2.14 all hold if we replace CPA *with* CCA. *In addition we have* $\mathsf{LoR\text{-}CCA} \not\Rightarrow \mathsf{\$\text{-}CPA}$ *and* $\mathsf{FtG\text{-}CCA} \not\to \mathsf{LoR\text{-}CPA}$.

Proof Note first that the decryption queries are the same in each notion. So we can extend each reduction by forwarding the decryption queries from the adversary we are reducing from to the oracle we are reducing to. The returned messages are distributed the same as in the original game and the bounds on stated advantages and consumed resources still hold.

The counterexample in Theorem 2.10 also holds if we allow decryption queries and also holds if we replace $\mathsf{FtG\text{-}CPA}$ with $\mathsf{FtG\text{-}CCA}$, but do not change $\mathsf{LoR\text{-}CPA}$. So we have $\mathsf{FtG\text{-}CCA} \not\to \mathsf{LoR\text{-}CPA}$.

The counterexample in the proof of Theorem 2.14 cannot be applied directly, as the scheme constructed there is insecure for any CCA game. We therefore construct a slightly different scheme. The encryption remains the same, adding n zero bits to the ciphertext. The decryption oracle now checks the first n bits and returns $\perp$ if they are not zero. If they are, then the remaining message is decrypted and returned. For an adversary in the $\mathsf{LoR\text{-}CCA}$ game against this new scheme we can construct an adversary against the original scheme. This adversary forwards encryption queries and adds 0^n to the ciphertext before returning them. Before forwarding decryption queries, it checks if the prefix 0^n is present. If it is, it is removed and the remaining ciphertext is forwarded. If it is not, $\perp$ is returned. This adversary simulates the game for the assumed adversary and has the same advantage. We can use the same adversary as in the proof of Theorem 2.14 to show that the new scheme not \$-CCA-secure. This construction also to shows $\mathsf{LoR\text{-}CCA} \not\Rightarrow \mathsf{\$\text{-}CPA}$. $\square$

We now show that the CCA notions games are indeed stronger than the CPA notions.

Theorem 2.18
Let $\mathsf{FL} \in \{\mathsf{SEM}, \mathsf{FtG}, \mathsf{LoR}, \mathsf{RoR}, \$\}$. *We have*

$$\mathsf{FL\text{-}CCA} \Rightarrow \mathsf{FL\text{-}CPA} \text{ and } \mathsf{\$\text{-}CPA} \not\Rightarrow \mathsf{FtG\text{-}CCA}.$$

Note that by the relations proved above, the second statement implies that for $F_1, F_2 \in \{\mathsf{LoR}, \mathsf{RoR}, \$, \mathsf{SEM}, \mathsf{FtG}\}$ we have $\mathsf{F_1\text{-}CPA} \not\Rightarrow \mathsf{F_2\text{-}CCA}$, so the CCA notions provide security in a way that no CPA notion can.

Proof Given an adversary for the $\mathsf{FL\text{-}CPA}$ game, we can clearly construct an adversary with equal advantage for the $\mathsf{FL\text{-}CCA}$ game by simply forwarding the queries.

For the second statement, consider a new encryption scheme which adds a random bit to the front of each ciphertext, which is ignored during decryption. For an adversary in the $-CPA game against this new scheme we can construct an adversary against the old scheme by simply adding and removing the random bit appropriately. This new adversary simulates the game for the assumed adversary and therefore has the same advantage.

On the other hand, this new scheme does not achieve FtG-CCA. For any challenge ciphertext we can construct a new ciphertext that decrypts to the same plaintext by flipping the first bit of a returned ciphertext. By Lemma 2.15 this allows the adversary to achieve advantage 1. $\square$

An overview containing the notions and relations of this chapter can be found in Figure 4.7.

Chapter 3

Stateful chosen-ciphertext notions

3.1 Stateful chosen-ciphertext security

Schemes which fulfil stateless correctness cannot prevent replay, reorder or dropping attacks, as the decryption algorithm must always output correct decryption for previously encrypted ciphertexts. So it important to analyse schemes which fulfil stateful correctness instead. Recall that such schemes only need to decrypt correctly if the sequence of decrypted ciphertexts is a subsequence of the sequence output by the encryption algorithm. This allows us to define stateful chosen-ciphertext privacy notions, which are stronger than the stateless notions. The original definition of stateful chosen-ciphertext security was proposed in [BKN02] using the flavor LoR. Many extensions on secure channels also use a LoR oracle, see for example [GM17] and [BDPS12]. A similar notion using the flavor \$ has also been defined, for example in [BDPS13].

For $\mathsf{FL} \in \{\mathsf{SEM}, \mathsf{FtG}, \mathsf{LoR}, \mathsf{RoR}\}$ it is fairly straightforward to define the notion FL-sfCCA. Instead of the set S, we initialise two counters i and j, which count the encryption and decryption queries, and a bit *sync*. The counters are used to notice when a mistake in the sequence of decrypted ciphertexts occurs. If such a mistake occurs, then the scheme is no longer bound by correctness, so we no longer need to suppress decryption queries. As long as there was no mistake we say the game is in-sync, after a mistake has occurred we call it out-of-sync. The bit *sync* to keeps track of this. In particular, a decryption query (and the ciphertext sent in this query) is called in-sync if at the end of the query $sync = 0$ and out-of-sync otherwise.

The encryption and challenge queries generally work the same as in the corresponding FL-CPA games, except that for each query where an encryption takes place we increase i and save the returned ciphertext as c_i. The decryption queries are somewhat different. First the input c is decrypted. We then update *sync* by checking if $j > i$, so if the adversary has decrypted more

Game $\mathsf{FL\text{-}sfCCA}_{\mathcal{SE}}(k, b)$

```
proc $\mathsf{Init}_{\mathsf{FL\text{-}sfCCA}}()$
    $i \leftarrow 0$
    $j \leftarrow 0$
    $sync \leftarrow 1$
    $k \leftarrow \mathsf{Init}_{\mathsf{FL\text{-}CPA}}()$
    return $k$

proc $\mathsf{Enc}_{\mathsf{FL\text{-}sfCCA}}(\overline{m})$
    $c \leftarrow \mathsf{Enc}_{\mathsf{FL\text{-}CPA}}(\overline{m})$
    if $c \neq \perp$
        $i \leftarrow i + 1$
        $c_i \leftarrow c$
    return $c$

proc $\mathsf{Dec}_{\mathsf{FL\text{-}sfCCA}}(c)$
    $j \leftarrow j + 1$
    $m \leftarrow \mathsf{Dec}_K(c)$
    if $j > i$ or $c \neq c_j$
        $sync \leftarrow 0$
    if $sync = 1$
        $m \leftarrow \perp$
    return $m$


proc $\mathsf{Fin}_{\mathsf{FL\text{-}sfCCA}}(\overline{d})$
    return $\mathsf{Fin}_{\mathsf{FL\text{-}CPA}}(\overline{d})$
```

```
proc $\mathsf{Init}_{\mathsf{\$\text{-}sfCCA}}()$
    $i \leftarrow 0$
    $j \leftarrow 0$
    $sync \leftarrow 1$
    $K \leftarrow \mathsf{KGen}_{\mathcal{SE}}(k)$
    return $k$

proc $\mathsf{Enc}_{\mathsf{\$\text{-}sfCCA}}(m)$
    $i \leftarrow i + 1$
    $c_i^{(0)} \leftarrow \{0, 1\}^{l(|m|)}$
    $c_i^{(1)} \leftarrow \mathsf{Enc}_K(m)$
    return $c_i^{(b)}$

proc $\mathsf{Dec}_{\mathsf{\$\text{-}sfCCA}}(c)$
    $j \leftarrow j + 1$
    if $j > i$ or $c \neq c_j^{(b)}$
        $sync \leftarrow 0$
    if $sync = 1$
        $m \leftarrow \perp$
        $\mathsf{Dec}_K(c_i^{(1)})$
    else
        $m \leftarrow \mathsf{Dec}_K(c)$
    return $m$

proc $\mathsf{Fin}_{\mathsf{\$\text{-}sfCCA}}(d)$
    return $d$
```

Figure 3.1: The game FL-sfCCA for FL $\in \{\mathsf{SEM}, \mathsf{FtG}, \mathsf{LoR}, \mathsf{RoR}, \$\}$.

than encrypted, or if $c \neq c_j$, so the ciphertext is not the next ciphertext in the order of encrypted ciphertexts. Note that once a game is out-of-sync it remains so. If the game is in-sync, then we return $\perp$. If it is out-of-sync, the actual output of the decryption is returned. In particular, an adversary can now decrypt a challenge ciphertexts as long as it is out-of-sync.

The decryption described above does not quite work for the flavor \$. If $b = 0$, then a random string is returned in encryption queries. If this random string is sent in an in-sync decryption query, passing it to the decryption algorithm would result in undetermined and out-of-sync behaviour. So for in-sync decryption queries we have to decrypt the ciphertext that was encrypted in the corresponding encryption query instead.

Definition 3.1 (FL-sfCCA)
For $\mathsf{FL} \in \{\mathsf{SEM}, \mathsf{FtG}, \mathsf{LoR}, \mathsf{RoR}, \$\}$ *the chosen-ciphertext security of an encryption scheme* $\mathcal{SE}$ *with security parameter* k *against adversaries with runtime* t *using* q *queries totalling* μ *bits is defined by* $\mathsf{Sec}^{\mathsf{FL\text{-}sfCCA}}_{\mathcal{SE}}(k, t, q, \mu)$ *using the game* $\mathsf{FL\text{-}sfCCA}_{\mathcal{SE}}(k, b)$ *depicted in Figure 3.1. The input for the queries are the same as in the corresponding* FL-CCA *game. The challenge queries are only defined if they exists in the corresponding* FL-CPA *game.*

Note that we could also define stateful chosen-plaintext notions. They are however equivalent to the stateless chosen-plaintext notions: If there are no decryption queries, then keeping track of the order of the encrypted messages does not change the game from the adversary's perspective.

3.2 Relations of stateful chosen-ciphertext notions

The following two theorems show that the stateful chosen-ciphertext notion is stronger than the normal chosen-ciphertext notion.

Theorem 3.2 (FL-sfCCA ⇒ FL-CCA)
Let $\mathcal{SE}$ *be a symmetric encryption scheme and* $\mathsf{FL} \in \{\mathsf{SEM}, \mathsf{FtG}, \mathsf{LoR}, \mathsf{RoR}, \$\}$ *. There exists a constant* c *so that for any* k*,* t*,* q *and* μ *we have*

$$\mathsf{Sec}^{\mathsf{FL\text{-}CCA}}_{\mathcal{SE}}(k, t, q, \mu) \leq \mathsf{Sec}^{\mathsf{FL\text{-}sfCCA}}_{\mathcal{SE}}(k, t + cq, q, \mu).$$

Proof Assume $\mathcal{A}_{\mathsf{FL\text{-}CCA}} \in \mathfrak{A}^{\mathsf{FL\text{-}CCA}}_{(t,q,\mu)}$ is an adversary. We construct an adversary $\mathcal{A}_{\mathsf{FL\text{-}sfCCA}} \in \mathfrak{A}^{\mathsf{FL\text{-}sfCCA}}_{(t+cq,q,\mu)}$ for an appropriate c. The new adversary runs $\mathcal{A}_{\mathsf{FL\text{-}CCA}}$ and is defined as follows, where $\overline{m}$ and $\overline{d}$ depend on the flavor and the challenge queries only apply to the appropriate flavors.

```
$\mathcal{A}_{\textsf{FL-sfCCA}}$
proc $\textsf{Init}_{\textsf{FL-CCA}}()$
    $S \leftarrow \varnothing$
    return $\textsf{Init}_{\textsf{FL-sfCCA}}()$

proc $\textsf{Enc}_{\textsf{FL-CCA}}(\overline{m})$
    $c \leftarrow \textsf{Enc}_{\textsf{FL-sfCCA}}(\overline{m})$
    $S \leftarrow S \bigcup c$
    return $c$

proc $\textsf{Chal}_{\textsf{FL-CCA}}(\overline{m})$
    $c \leftarrow \textsf{Chal}_{\textsf{FL-sfCCA}}(\overline{m})$
    $S \leftarrow S \bigcup c$
    return $c$

proc $\textsf{Dec}_{\textsf{FL-CCA}}(c)$
    if $c \in S$
        $m \leftarrow \perp$
    else
        $m \leftarrow \textsf{Dec}_{\textsf{FL-sfCCA}}(c)$
    return $m$

proc $\textsf{Fin}_{\textsf{FL-CCA}}(\overline{d})$
    return $\textsf{Fin}_{\textsf{FL-sfCCA}}(\overline{d})$
```

The number of queries and their bitsize are the same. The difference in runtime consists of the updating and checking of S in $\mathcal{A}_{\textsf{FL-sfCCA}}$ and of i and j in $\textsf{Enc}_{\textsf{FL-sfCCA}}$ and $\textsf{Dec}_{\textsf{FL-sfCCA}}$. This can be bounded by a constant c for each query.

As all in-sync decryption queries are contained in S, any decryption query that is sent to the decryption oracle is out-of-sync, and therefore the decryption is returned. Any decryption of a ciphertext not contained in S would be out-of-sync in a FL-CCA game as well, so $\mathcal{A}_{\textsf{FL-sfCCA}}$ simulates the game $\textsf{FL-CCA}_{\mathcal{SE}}(k, b)$ for the value of b as in the actual FL-sfCCA game and therefore has the same advantage as $\mathcal{A}_{\textsf{FL-CCA}}$, as desired. □

For the converse statement we prove a stronger result, namely that the stateful notions are not attainable by schemes which fulfil the stateless correctness notion. This implies that $\textsf{F}_1\textsf{-CCA} \not\Rightarrow \textsf{F}_2\textsf{-sfCCA}$ for any two flavors F_1 and F_2, under the assumption that there exists a scheme $\mathcal{SE}$ which fulfils stateless correctness so that $\textsf{Sec}_{\mathcal{SE}}^{\textsf{F}_1\textsf{-CCA}}$ is small.

Theorem 3.3
Let $\mathcal{SE}$ be a scheme which fulfils the stateless correctness notion. There exist t and μ so that

$$\textsf{Sec}_{\mathcal{SE}}^{\textsf{FtG-sfCCA}}(k, t, 3, \mu) = 1.$$

Proof Consider an adversary that sends a challenge query with two different messages and then sends two decryption queries with the returned ciphertext. The message space and the runtime of these queries determine the values of t and μ, but they are fairly small. The first decryption query is in-sync, so $\perp$ is returned. The second query is out-of-sync, so its decryption is returned. As the scheme fulfils stateless correctness, this is the actual decryption. By Lemma 2.15, this allows the adversary to achieve advantage 1. $\square$

We actually do not need to assume stateless correctness to prove this theorem. It suffices to assume that there exists some sequence of encryptions and decryptions so that an out-of-sync decryption query leaks information about the plaintext. We will take a closer look at interplay of correctness and privacy in Chapter 7.

3.3 Relations between stateful notions

Theorem 3.4 (Relations of LoR-, RoR-, FtG-, SEM- and \$-sfCCA)
The Theorems 2.6, 2.7, 2.8, 2.9, 2.10, 2.11, 2.12, 2.13 and 2.14 hold if we replace CPA *with* sfCCA. *Also we have* FtG-sfCCA $\rightarrow$ LoR-CCA *and* LoR-sfCCA $\not\rightarrow$ \$-CCA.

Proof Similar to the proof of Theorem 2.17, we can forward the decryption queries in each reduction to create a reduction for the sfCCA-notions of each flavor. The claimed resources and bounds on advantage are not affected by this. The schemes used as counterexamples in Theorem 2.17 can also be applied to stateful schemes and also show the last two statements. $\square$

An overview of the notion and relations of this chapter can be found in Figure 4.7.

Chapter 4

Integrity and authenticated encryption

4.1 Integrity

The notions defined so far only ensure the privacy of encryption schemes. We now wish to also consider their integrity. Integrity ensures that an adversary cannot forge ciphertexts. To model attacks on the integrity of encryption schemes we define games in which an adversary with access to a truthful encryption oracle attempts to forge a valid ciphertext.

For schemes which fulfil stateless correctness we define two notions of integrity, plaintext integrity INT-CTXT and ciphertext integrity INT-PTXT, following the definitions from [BN00]. For ciphertext integrity, the adversary simply has to forge a new ciphertext, for plaintext integrity it has to forge a ciphertext for a message which was not encrypted yet. Similarly, we define two notions for the stateful correctness, INT-sfCTXT and INT-sfPTXT. In the first, originally defined in [BKN02], the adversary has to send an out-of-sync decryption query with a valid ciphertext. In the second, originally defined for a somewhat different scenario in [BSWW13], the adversary has to send a ciphertext that decrypts to an out-of-sync plaintext. These games do not have a bit b, so the advantage of adversaries is defined slightly differently. Also these games do not have flavors of encryption, so they do not fit in to the naming scheme used so far. Instead we use their classical names.

Definition 4.1 (INT-(sf)CTXT/(sf)PTXT)
For $\mathsf{X} \in \{\mathsf{CTXT}, \mathsf{PTXT}, \mathsf{sfCTXT}, \mathsf{sfPTXT}\}$ *let*

$$\mathsf{Adv}^{\mathsf{INT\text{-}X}}_{\mathcal{SE},\mathcal{A}}(k) = \Pr[\mathsf{INT\text{-}X}^{\mathcal{A}}_{\mathcal{SE}}(k) = 1]$$

where $\mathsf{INT\text{-}X}^{\mathcal{A}}_{\mathcal{SE}}(k)k$ *is the outcome of the game* $\mathsf{INT\text{-}X}_{\mathcal{SE}}(k)$ *depicted in Figure 4.1. The input for encryption queries are messages* $m \in \mathcal{M}$ *and for decryption queries strings* c.

```
Game INT-CTXT_{SE}(k)
proc $\mathsf{Init}_{\mathsf{INT\text{-}CTXT}}()$
    $K \leftarrow\!\$\ \mathsf{KGen}(k)$
    $win \leftarrow 0$
    $S \leftarrow \varnothing$
    return $k$

proc $\mathsf{Enc}_{\mathsf{INT\text{-}CTXT}}(m)$
    $c \leftarrow \mathsf{Enc}_K(m)$
    $S \leftarrow S \cup \{c\}$
    return $c$

proc $\mathsf{Dec}_{\mathsf{INT\text{-}CTXT}}(c)$
    $m \leftarrow \mathsf{Dec}_K(c)$
    if $m \neq \perp$ and $c \notin S$
        $win \leftarrow 1$
    return $m \neq \perp$

proc $\mathsf{Fin}_{\mathsf{INT\text{-}CTXT}}()$
    return $win$
```

```
Game INT-sfCTXT_{SE}(k)
proc $\mathsf{Init}_{\mathsf{INT\text{-}sfCTXT}}()$
    $K \leftarrow\!\$\ \mathsf{KGen}(k)$
    $win \leftarrow 0$
    $i \leftarrow 0$
    $j \leftarrow 0$
    $sync \leftarrow 1$
    return $k$

proc $\mathsf{Enc}_{\mathsf{INT\text{-}sfCTXT}}(m)$
    $i \leftarrow i + 1$
    $m_i \leftarrow m$
    $c_i \leftarrow \mathsf{Enc}_K(m)$
    return $c_i$

proc $\mathsf{Dec}_{\mathsf{INT\text{-}sfCTXT}}(c)$
    $j \leftarrow j + 1$
    $m \leftarrow \mathsf{Dec}_K(c)$
    if $j > i$ or $c \neq c_j$
        $sync \leftarrow 0$
    if $m \neq \perp$ and $sync = 0$
        $win \leftarrow 1$
    return $m \neq \perp$

proc $\mathsf{Fin}_{\mathsf{INT\text{-}sfCTXT}}()$
    return $win$
```

Figure 4.1: The games for the notions INT-CTXT and INT-sfCTXT. The game for INT-PTXT is defined by replacing the line 12 of the INT-CTXT game with **if** $m \neq \perp$ and $m \notin S$. The game for INT-sfPTXT is defined by replacing the line 16 of the INT-sfCTXT game with **if** $j > i$ or $m \neq m_j$.

The integrity of an encryption scheme $\mathcal{SE}$ with security parameter k against adversaries with runtime t using q queries totalling μ bits is defined by $\mathsf{Sec}^{\mathsf{INT\text{-}X}}_{\mathcal{SE}}(k, t, q, \mu)$.

Similarly to the privacy notions, the stateful integrity notions imply the stateless version.

Theorem 4.2 (INT-sfPTXT ⇒ INT-PTXT, INT-sfCTXT ⇒ INT-CTXT)
Let $\mathcal{SE}$ be a symmetric encryption scheme. There exists a constant c so that for any k, t, q and μ we have

$$\mathsf{Sec}^{\mathsf{INT\text{-}PTXT}}_{\mathcal{SE}}(k, t, q, \mu) \leq \mathsf{Sec}^{\mathsf{INT\text{-}sfPTXT}}_{\mathcal{SE}}(k, t + cq, q, \mu).$$

$$\mathsf{Sec}^{\mathsf{INT\text{-}CTXT}}_{\mathcal{SE}}(k, t, q, \mu) \leq \mathsf{Sec}^{\mathsf{INT\text{-}sfCTXT}}_{\mathcal{SE}}(k, t + cq, q, \mu).$$

Proof For the first statement assume we have an adversary $\mathcal{A}_{\mathsf{INT\text{-}PTXT}}$ for the $\mathsf{INT\text{-}PTXT}$ game. We define an adversary $\mathcal{A}_{\mathsf{INT\text{-}sfPTXT}}$ for the $\mathsf{INT\text{-}sfPTXT}$ game that runs $\mathcal{A}_{\mathsf{INT\text{-}PTXT}}$ and simply forwards its queries. This simulates the $\mathsf{INT\text{-}PTXT}$ game. Consider the query (if any) which would set *win* to 1 in the simulated $\mathsf{INT\text{-}PTXT}$ game. The decryption of the ciphertext sent in this query was never encrypted. In particular, it was not the j-th message encrypted, so *sync* is set to 0 and win is set to true in the $\mathsf{INT\text{-}sfPTXT}$ game as well. The switch in games results in a constant runtime increase per query.

The same argument holds for the $\mathsf{INT\text{-}CTXT}$ and $\mathsf{INT\text{-}sfCTXT}$ games. $\square$

Next we show that ciphertext integrity is stronger than plaintext integrity.

Theorem 4.3 ($\mathsf{INT\text{-}(sf)CTXT} \Rightarrow \mathsf{INT\text{-}(sf)PTXT}$)
Let $\mathcal{SE}$ be a symmetric encryption scheme. For any k, t, q and μ we have

$$\mathsf{Sec}_{\mathcal{SE}}^{\mathsf{INT\text{-}(sf)PTXT}}(k,t,q,\mu) \leq \mathsf{Sec}_{\mathcal{SE}}^{\mathsf{INT\text{-}(sf)CTXT}}(k,t,q,\mu).$$

On the other hand we have $\mathsf{INT\text{-}sfPTXT} \not\Rightarrow \mathsf{INT\text{-}CTXT}$*, so there exists an encryption scheme $\mathcal{SE}'$ and a constant c so that*

$$\begin{aligned} \mathsf{Sec}_{\mathcal{SE}'}^{\mathsf{INT\text{-}sfPTXT}}(k,t,q,\mu) &= \mathsf{Sec}_{\mathcal{SE}}^{\mathsf{INT\text{-}sfPTXT}}(k,t+cq,q,\mu) \\ \mathsf{Sec}_{\mathcal{SE}'}^{\mathsf{INT\text{-}CTXT}}(k,t,2,\mu) &= 1 \end{aligned}$$

Proof Assume $\mathcal{A}_{\mathsf{INT\text{-}(sf)PTXT}} \in \mathfrak{A}_{(t,q,\mu)}^{\mathsf{INT\text{-}(sf)PTXT}}$ is an adversary and define an adversary $\mathcal{A}_{\mathsf{INT\text{-}(sf)CTXT}} \in \mathfrak{A}_{(t,q,\mu)}^{\mathsf{INT\text{-}(sf)CTXT}}$ that runs $\mathcal{A}_{\mathsf{INT\text{-}(sf)PTXT}}$ and simply forwards its queries. This adversary uses the stated resources and simulates the $\mathsf{INT\text{-}(sf)PTXT}$ game. Now consider the query (if any) which would set *win* to 1 in the simulated $\mathsf{INT\text{-}(sf)PTXT}$ game. We now differentiate between the stateless and stateful scenario. In the stateless scenario the ciphertext sent in this query must be new, as otherwise (by stateless correctness) it would not decrypt to a new message. So this query also sets *win* to 1 in the $\mathsf{INT\text{-}CTXT}$ game. In the stateful scenario, note that (by stateful correctness) $m \neq m_j$ implies that $c \neq c_j$, so this query is out-of-sync and *win* is also set to 1 in the $\mathsf{INT\text{-}(sf)CTXT}$ game. So we have

$$\mathsf{Adv}_{\mathcal{SE},\mathcal{A}_{\mathsf{INT\text{-}PTXT}}}^{\mathsf{INT\text{-}PTXT}}(k) \leq \mathsf{Adv}_{\mathcal{SE},\mathcal{A}_{\mathsf{INT\text{-}CTXT}}}^{\mathsf{INT\text{-}CTXT}}(k),$$

as desired. For the separation consider the scheme $\mathcal{SE}'$ defined in Theorem 2.18 which adds a random bit to the front of each ciphertext. For an adversary in the $\mathsf{INT\text{-}sfPTXT}$ against $\mathcal{SE}'$ we can construct an adversary against $\mathcal{SE}$ in the same game, which adds and removes this bit appropriately. This simulates the game against $\mathcal{SE}'$ and uses the stated resources, so the first statement holds. For the second consider an adversary for the $\mathsf{INT\text{-}CTXT}$

game which encrypts an arbitrary message and then decrypts the same ciphertext with the first bit flipped. This ciphertext is new and decrypts to original message, so the adversary wins. The adversary uses the claimed resources. Note that this argument combined with Theorem 4.2 also shows that $\mathsf{INT\text{-}sfPTXT} \not\Rightarrow \mathsf{INT\text{-}sfCTXT}$ and $\mathsf{INT\text{-}PTXT} \not\Rightarrow \mathsf{INT\text{-}sfCTXT}$. □

Similarly to Theorem 3.3, the stateful notions of integrity cannot be achieved by schemes which fulfil stateless correctness.

Theorem 4.4
Let $\mathcal{SE}$ be an encryption scheme which fulfils stateless correctness. There exist t and μ so that

$$\mathsf{Sec}^{\mathsf{INT\text{-}sfPTXT}}_{\mathcal{SE}}(k, t, 5, \mu) = 1.$$

Proof Consider an adversary which encrypts an arbitrary message and then decrypts the returned ciphertext twice. The second decryption query sets *win* to 1. The message space and the runtime of these queries determine the values of t and μ. □

This theorem also implies (under the assumption that a scheme with stateless integrity exist) that stateful integrity does not only imply stateless integrity as we saw above, but is indeed stronger. Specifically, the following relation holds:

$$\begin{aligned}
\mathsf{INT\text{-}PTXT} &\not\Rightarrow \mathsf{INT\text{-}sfPTXT}\\
\mathsf{INT\text{-}CTXT} &\not\Rightarrow \mathsf{INT\text{-}sfCTXT}\\
\mathsf{INT\text{-}CTXT} &\not\Rightarrow \mathsf{INT\text{-}sfPTXT}.
\end{aligned}$$

As the next two theorems show, the notions of integrity are not comparable with the notion defined so far.

Theorem 4.5 ($-sfCCA $\not\Rightarrow$ INT-PTXT)
Let $\mathcal{SE}$ be a symmetric encryption scheme. There exists an encryption scheme $\mathcal{SE}'$ so that for any k, t, q and μ we have

$$\begin{aligned}
\mathsf{Sec}^{\mathsf{\$\text{-}sfCCA}}_{\mathcal{SE}'}(k, t, q, \mu) &= \mathsf{Sec}^{\mathsf{\$\text{-}sfCCA}}_{\mathcal{SE}}(k, t, q, \mu)\\
\mathsf{Sec}^{\mathsf{INT\text{-}PTXT}}_{\mathcal{SE}'}(k, t, 1, \mu) &= 1
\end{aligned}$$

Proof Consider the following scheme $\mathcal{SE}'$

Scheme $\mathcal{SE}'$

```
proc KGen_{SE'}(k)
    K ← KGen_{SE}(k)
    return K

proc Enc_{SE'}(K, m)
    c ← Enc_{SE}(K, m)
    return c

proc Dec_{SE'}(K, c)
    m = Dec_{SE}(K, c)
    if m = ⊥
        m ← 0
    return m
```

An adversary in the INT-PTXT game can simply query an arbitrary string for decryption. As the decryption algorithm of $\mathcal{SE}'$ never outputs $\perp$, this adversary always wins. On the other hand consider an adversary in the \$-sfCCA game against $\mathcal{SE}'$. We can construct an adversary against $\mathcal{SE}$ by forwarding encryption queries and replacing $\perp$ with 0 in the answers to decryption queries. This simulates the game against $\mathcal{SE}'$ with the same value of b, so the advantage is preserved. $\square$

Theorem 4.6 (INT-sfCTXT $\not\Rightarrow$ FtG-CPA)
Let $\mathcal{SE}$ be a symmetric encryption scheme and $\mathsf{FL} \in \{\mathsf{SEM}, \mathsf{FtG}, \mathsf{LoR}, \mathsf{RoR}, \$\}$ *. There exists an encryption scheme $\mathcal{SE}'$ so that for any k, t, q and μ we have*

$$\mathsf{Sec}^{\mathsf{INT\text{-}sfCTXT}}_{\mathcal{SE}'}(k,t,q,\mu) = \mathsf{Sec}^{\mathsf{INT\text{-}sfCTXT}}_{\mathcal{SE}}(k,t+cq,q,\mu)$$
$$\mathsf{Sec}^{\mathsf{FL\text{-}CPA}}_{\mathcal{SE}'}(k,t,q,\mu) = 1$$

Proof Consider the following scheme:

Scheme $\mathcal{SE}'$

```
proc KGen_{SE'}(k)
    K ←$ KGen_{SE}(k)
    return K

proc Enc_{SE'}(K, m)
    c ← Enc_{SE}(K, m)
    c' ← (c, m)
    return c'

proc Dec_{SE'}(K, c')
    (c, m) ← c'
    m' ← Dec_{SE}(K, c)
    if m' ≠ m
        m' ← ⊥
    return m'
```

Assuming we have an adversary for the INT-(sf)CTXT game against $\mathcal{SE}'$ we can construct an adversary against $\mathcal{SE}$ which uses the claimed resources that adds, removes and checks the message itself. This simulates the game against $\mathcal{SE}'$, so the first statement holds. By Lemma 2.15 we have

$$\mathsf{Sec}^{\mathsf{FtG\text{-}CPA}}_{\mathcal{SE}'}(k,t,q,\mu) = 1,$$

implying the second statement. $\square$

$$\begin{tikzcd}
 & \boxed{\mathsf{INT\text{-}CTXT}} \arrow[r, Rightarrow, shift right, "/" marking] \arrow[d, Rightarrow, shift right] \arrow[dr, Leftrightarrow, "/" marking] & \boxed{\mathsf{INT\text{-}sfCTXT}} \arrow[l, Rightarrow, shift right] \arrow[r, Rightarrow, "/" marking] \arrow[d, Rightarrow, shift right] & \boxed{\mathsf{FtG\text{-}CPA}} \\
\boxed{\$'\mathsf{\text{-}sfCCA}} \arrow[r, Rightarrow, "/" marking] & \boxed{\mathsf{INT\text{-}PTXT}} \arrow[u, Rightarrow, shift right, "/" marking] \arrow[r, Rightarrow, shift right, "/" marking] & \boxed{\mathsf{INT\text{-}sfPTXT}} \arrow[u, Rightarrow, shift right, "/" marking] \arrow[l, Rightarrow, shift right] &
\end{tikzcd}$$

Figure 4.2: An overview of the relations between notions of integrity.

4.2 Authenticated encryption

We have now established privacy and integrity as two separate goals for encryption schemes. Of course we are actually interested in schemes which provide both. Such schemes are often denoted authenticated encryption schemes, first introduced in [BN00] as the combination of LoR-CPA and INT-CTXT. This combination is shown there to imply LoR-CCA, a result known as generic composition. Similarly, authenticated encryption for stateful schemes was originally introduced as the combination of LoR-CPA and INT-sfCTXT and shown to imply LoR-sfCCA in [BKN02].

In [Shr04] a combined notion IND-CCA3 is introduced and shown to be equivalent to the combination of RoR-CPA and a version of INT-CTXT. In [RS06] a combined game was defined using a \$-type oracle. For stateful schemes a combined game using a LoR oracle was defined in [BHMS15]. Interestingly, to our knowledge no stateful authenticated encryption notion has been defined using a \$-type oracle. This is probably due to the fact that most stateful schemes defined in the past have contained some sort of header information which is distinguishable from random noise. We now define such a notion and will see in Chapter 6 that TLS 1.3 actually achieves it.

The idea behind this combined notion is that the adversary can win the game if it breaks either the integrity or the privacy of the scheme. This is achieved by taking the FL-(sf)CCA game and replacing the truthful decryption oracle with a challenge decryption oracle, which returns $\perp$ if $b = 0$. If the adversary can forge a valid ciphertext, than it can distinguish whether decryption queries are being suppressed.

Definition 4.7 (FL-(sf)AE)
For FL $\in \{\mathsf{SEM}, \mathsf{FtG}, \mathsf{LoR}, \mathsf{RoR}, \$\}$ *the authenticated encryption security of an encryption scheme* $\mathcal{SE}$ *with security parameter* k *against adversaries with runtime* t *using* q *queries totalling* μ *bits is defined by* $\mathsf{Sec}_{\mathcal{SE}}^{\mathsf{FL\text{-}(sf)AE}}(k, t, q, \mu)$ *using the game* $\mathsf{FL\text{-}(sf)AE}_{\mathcal{SE}}(k, b)$ *depicted in Figure 4.3. The input for the queries are the same as in the corresponding* FL-CCA *game. The challenge queries are only defined if they exists in the corresponding* FL-CPA *game.*

```
Game FL-(sf)AE$_{\mathcal{SE}}(k, b)$

proc Init$_{\text{FL-(sf)AE}}()$


    return Init$_{\text{FL-(sf)CCA}}()$

proc Enc$_{\text{FL-(sf)AE}}(\overline{m})$
    $c \leftarrow$ Enc$_{\text{FL-(sf)CCA}}(\overline{m})$
    return $c$

proc Chal$_{\text{FL-(sf)AE}}(\overline{m})$
    $c \leftarrow$ Chal$_{\text{FL-(sf)CCA}}(\overline{m})$
    return $c$

proc Dec$_{\text{FL-(sf)AE}}(c)$
    if $b = 0$
        $m \leftarrow \perp$
    else
        $m \leftarrow$ Dec$_{\text{FL-(sf)CCA}}(c)$
    return $m$

proc Fin$_{\text{FL-(sf)AE}}(\overline{d})$
    return Fin$_{\text{FL-(sf)CCA}}(\overline{d})$
```

```
Game \$-AE$_{\mathcal{SE}}(k, b)$

proc Init$_{\text{\$-AE}}()$
    $S \leftarrow \varnothing$
    $K \leftarrow\$$ KGen$(k)$
    return $k$

proc Enc$_{\text{\$-AE}}(m)$
    $c_0 \leftarrow\$ \{0,1\}^{l(|m|)}$
    $c_1 \leftarrow$ Enc$_K(m)$
    $S \leftarrow S \cup \{c_b\}$
    return $c_b$


proc Dec$_{\text{\$-AE}}(c)$
    if $b = 0$ or $c \in S$
        $m \leftarrow \perp$
    else
        $m \leftarrow$ Dec$_K(c)$
    return $m$

proc Fin$_{\text{\$-AE}}(d)$
    return $d$
```

Figure 4.3: The games for the notions FL-(sf)AE and \$-AE.

Our first question is whether the relations between the different flavors are the same. Indeed, the following theorem shows that they are.

Theorem 4.8 (Relations of LoR-, RoR-, FtG-, SEM- and \$-(sf)AE)
*The Theorems 2.6 (*RoR-CPA $\Rightarrow$ LoR-CPA*), 2.7 (*LoR-CPA $\Rightarrow$ RoR-CPA*), 2.8 (*LoR-CPA $\Rightarrow$ FtG-CPA*), 2.9 + 2.10 (*FtG-CPA $\rightarrow$ LoR-CPA*), 2.11 (*SEM-CPA $\Rightarrow$ FtG-CPA*), 2.12 (*FtG-CPA $\Rightarrow$ SEM-CPA*), 2.13 (*\$-CPA $\Rightarrow$ LoR-CPA*) and 2.14 (*LoR-CPA $\not\Rightarrow$ \$-CPA*) hold if we replace* CPA *with* AE *or with* sfAE*.*

Also we have FtG-(sf)AE $\rightarrow$ LoR-(sf)CCA, FtG-sfAE $\rightarrow$ LoR-AE, LoR-(sf)AE $\not\Rightarrow$ \$-(sf)CCA *and* LoR-sfAE $\not\Rightarrow$ \$-AE*.*

Proof As in Theorem 2.17 and Theorem 3.4, we can forward the decryption queries to prove the reductions. The counterexample of Theorem 2.14 holds if we use the scheme constructed in Theorem 2.17. The counterexample of Theorem 2.10 however does not hold any more, as for any message m the ciphertext $0 \| m$ is now a valid ciphertext. This allows an adversary to easily check whether the decryption oracle is actually decrypting messages or just returning $\perp$.

Instead we define a new scheme $\mathcal{SE}'$. This scheme still leaks the encrypted message for one randomly chosen encryption query, but does it in a way that preserves integrity. Specifically, the scheme now appends a bit b to the message before encrypting using the encryption of $\mathcal{SE}$. For the random query that leaks the encrypted message $b = 1$, otherwise $b = 0$. The ciphertexts in this scheme consists of b, the ciphertext and possibly the message. The decryption algorithm checks that the bit matches the first bit of the decrypted ciphertext. This ensures that an adversary cannot generate a valid new ciphertext by flipping the bit, as it would have to generate the appropriate ciphertext for the message with first bit flipped.

Scheme $\mathcal{SE}'$

```
proc $\mathsf{KGen}_{\mathcal{SE}'}(k)$
  $K \leftarrow\$ \mathsf{KGen}_{\mathcal{SE}}(k)$
  $i \leftarrow 0$
  $j \leftarrow\$ \{1, \ldots, q\}$
  return $(K, (i, j))$

proc $\mathsf{Enc}_{\mathcal{SE}'}(K, m, (i, j))$
  $i \leftarrow i + 1$
  if $j = i$
    $c \leftarrow \mathsf{Enc}_{\mathcal{SE}}(K, 1 \| m)$
    $c' \leftarrow (1, c, m)$
  else
    $c \leftarrow \mathsf{Enc}_{\mathcal{SE}}(K, 0 \| m)$
    $c' \leftarrow (0, c, 0)$
  return $(c', (i, j))$

proc $\mathsf{Dec}_{\mathcal{SE}'}(K, c')$
  $(b, c, m) \leftarrow c'$
  $m' \leftarrow \mathsf{Dec}_{\mathcal{SE}}(K, c)$
  if $b = 1$ and $m' \neq m$
    $m' \leftarrow \perp$
  return $m'$
```

An adversary in the LoR-(sf)AE game which queries two different messages q times can achieve advantage 1, as the j-th query leaks which message was encoded.

On the other hand, assume we have an adversary for the FtG-(sf)AE game against $\mathcal{SE}'$. We can construct an adversary against $\mathcal{SE}$ that chooses j itself and simulates $\mathcal{SE}'$. This only fails if the j-th encryption query is the challenge query. This is now the same scenario as in Theorem 2.10 and the remaining proof follows the same way. □

Next we show that the AE notions imply the CCA notions of the same flavor.

Theorem 4.9 ($\mathsf{FL\text{-}(sf)AE} \Rightarrow \mathsf{FL\text{-}(sf)CCA}$)
Let $\mathcal{SE}$ be a symmetric encryption scheme and $\mathsf{FL} \in \{\mathsf{SEM}, \mathsf{FtG}, \mathsf{LoR}, \mathsf{RoR}, \$\}$. There exists a constant c so that for any k, t, q and μ we have

$$\mathsf{Sec}_{\mathcal{SE}}^{\mathsf{FL\text{-}(sf)CCA}}(k, t, q, \mu) \leq 2 \cdot \mathsf{Sec}_{\mathcal{SE}}^{\mathsf{FL\text{-}(sf)AE}}(k, t + cq, q, \mu).$$

Proof This proof is based on the ideas from Theorem 3.1 [BN00], which considers the stateless case and the flavor LoR and Proposition 6.4 from [BKN02], which considers the stateful version of LoR. Note that because of Theorem 2.17, Theorem 3.4 and Theorem 4.8, it suffices to prove the statement for $\mathsf{FL} \in \{\mathsf{LoR}, \mathsf{FtG}, \$\}$. Consider an adversary $\mathcal{A}_{\mathsf{FL\text{-}(sf)CCA}} \in \mathfrak{A}_{(t,q,\mu)}^{\mathsf{FL\text{-}(sf)CCA}}$. Note that the games $\mathsf{FL\text{-}(sf)AE}_{\mathcal{SE}}(k, 1)$ and $\mathsf{FL\text{-}(sf)CCA}_{\mathcal{SE}}(k, 1)$ are the same. We now define a game $\mathsf{FL^0\text{-}(sf)AE}$, which has the same queries as $\mathsf{FL\text{-}(sf)AE}$. Encryption and challenge queries however do not depend on the value of b, instead they are always answered as if $b = 0$. Additionally, decryption queries are reversed, so for $b = 0$ they are answered, for $b = 1$ $\perp$ is returned. Note that the game $\mathsf{FL^0\text{-}(sf)AE}_{\mathcal{SE}}(k, 1)$ is equal to the game $\mathsf{FL\text{-}(sf)AE}_{\mathcal{SE}}(k, 0)$ and similarly the game $\mathsf{FL^0\text{-}(sf)AE}_{\mathcal{SE}}(k, 0)$ is equal to the game $\mathsf{FL\text{-}(sf)CCA}_{\mathcal{SE}}(k, 0)$. So we have

$$\begin{aligned}
&\mathsf{Adv}_{\mathcal{SE},\mathcal{A}_{\mathsf{FL\text{-}(sf)CCA}}}^{\mathsf{FL\text{-}(sf)CCA}}(k) \\
&= \Pr[\mathsf{FL\text{-}(sf)CCA}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{FL\text{-}(sf)CCA}}}(k, 1) = 1] - \Pr[\mathsf{FL\text{-}(sf)CCA}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{FL\text{-}(sf)CCA}}}(k, 0) = 1] \\
&= \mathsf{Adv}_{\mathcal{SE},\mathcal{A}_{\mathsf{FL\text{-}(sf)CCA}}}^{\mathsf{FL\text{-}(sf)AE}}(k) + \mathsf{Adv}_{\mathcal{SE},\mathcal{A}_{\mathsf{FL\text{-}(sf)CCA}}}^{\mathsf{FL^0\text{-}(sf)AE}}(k).
\end{aligned}$$

We now construct an adversary $\mathcal{A}_{\mathsf{FL\text{-}(sf)AE}}$ which runs $\mathcal{A}_{\mathsf{FL\text{-}(sf)CCA}}$ and simulates the $\mathsf{FL^0\text{-}(sf)AE}$ game. The goal of this adversary is to always simulate encryption queries for $b = 0$. This is fairly straightforward for $\mathsf{FL} \in \{\mathsf{LoR}, \mathsf{FtG}\}$, as the adversary can send (m_0, m_0) to ensure that m_0 is encrypted. For \$ we have to be somewhat more careful. To ensure that a random ciphertext is returned, the adversary samples one itself. For the stateless version, the adversary does not even have to call $\mathsf{Enc}_{\mathsf{\$\text{-}AE}}$. Decryption queries containing these random strings are ignored, as expected by $\mathcal{A}_{\mathsf{\$\text{-}CCA}}$. For the stateful version, the adversary has to call $\mathsf{Enc}_{\mathsf{\$\text{-}sfAE}}$ to ensure that the state is progressed. It still samples an additional random string to return to $\mathcal{A}_{\mathsf{\$\text{-}sfCCA}}$ and ensures that the correct ciphertext is decrypted for in-sync decryption queries. Finally, the output of $\mathcal{A}_{\mathsf{FL\text{-}(sf)CCA}}$ is reversed to capture the reversed decryption oracle in the $\mathsf{FL^0\text{-}(sf)AE}$ game. The adversaries for each flavor can be found in Figure 4.4. They use the same number of queries with the same size. The runtime of each query is only increased by constant operations, note in particular that the random sampling of a message in the $\mathsf{Enc}_{\mathsf{\$\text{-}(sf)CCA}}$ queries happens in $\mathcal{A}_{\mathsf{FL\text{-}(sf)AE}}$ instead of, not in addition to, in the $\mathsf{\$\text{-}(sf)CCA}$ game. So the claimed resources are used.

$\mathcal{A}_{\mathsf{LoR\text{-}(sf)AE}}$

```
proc $\mathsf{Init}_{\mathsf{LoR\text{-}(sf)CCA}}()$
  return $\mathsf{Init}_{\mathsf{LoR\text{-}(sf)AE}}()$

proc $\mathsf{Enc}_{\mathsf{LoR\text{-}(sf)CCA}}(m_0, m_1)$
  $c \leftarrow \mathsf{Enc}_{\mathsf{LoR\text{-}(sf)AE}}(m_0, m_0)$
  return $c$

proc $\mathsf{Dec}_{\mathsf{LoR\text{-}(sf)CCA}}(c)$
  $m \leftarrow \mathsf{Dec}_{\mathsf{LoR\text{-}(sf)AE}}(c)$
  return $m$

proc $\mathsf{Fin}_{\mathsf{LoR\text{-}(sf)CCA}}(d)$
  return $\mathsf{Fin}_{\mathsf{LoR\text{-}(sf)AE}}(1-d)$
```

$\mathcal{A}_{\mathsf{FtG\text{-}(sf)AE}}$

```
proc $\mathsf{Init}_{\mathsf{FtG\text{-}(sf)CCA}}()$
  return $\mathsf{Init}_{\mathsf{FtG\text{-}(sf)AE}}()$

proc $\mathsf{Enc}_{\mathsf{FtG\text{-}(sf)CCA}}(m)$
  $c \leftarrow \mathsf{Enc}_{\mathsf{FtG\text{-}(sf)AE}}(m)$
  return $c$

proc $\mathsf{Chal}_{\mathsf{FtG\text{-}(sf)CCA}}(m_0, m_1)$
  $c \leftarrow \mathsf{Enc}_{\mathsf{FtG\text{-}(sf)AE}}(m_0, m_0)$
  return $c$

proc $\mathsf{Dec}_{\mathsf{FtG\text{-}(sf)CCA}}(c)$
  $m \leftarrow \mathsf{Dec}_{\mathsf{FtG\text{-}(sf)AE}}(c)$
  return $m$

proc $\mathsf{Fin}_{\mathsf{FtG\text{-}(sf)CCA}}(d)$
  return $\mathsf{Fin}_{\mathsf{FtG\text{-}(sf)AE}}(1-d)$
```

$\mathcal{A}_{\mathsf{\$\text{-}AE}}$

```
proc $\mathsf{Init}_{\mathsf{\$\text{-}CCA}}()$
  $S \leftarrow \varnothing$
  return $\mathsf{Init}_{\mathsf{\$\text{-}AE}}()$

proc $\mathsf{Enc}_{\mathsf{\$\text{-}CCA}}(m)$
  $c \leftarrow\$ \{0,1\}^{l(|m|)}$
  $S \leftarrow S \cup c$
  return $c'$

proc $\mathsf{Dec}_{\mathsf{\$\text{-}CCA}}(c)$
  if $c \in S$
    $m \leftarrow \perp$
  else
    $m \leftarrow \mathsf{Dec}_{\mathsf{\$\text{-}AE}}(c)$
  return $m$

proc $\mathsf{Fin}_{\mathsf{\$\text{-}CCA}}(d)$
  return $\mathsf{Fin}_{\mathsf{\$\text{-}AE}}(1-d)$
```

$\mathcal{A}_{\mathsf{\$\text{-}sfAE}}$

```
proc $\mathsf{Init}_{\mathsf{\$\text{-}sfCCA}}()$
  $i \leftarrow 0$
  $j \leftarrow 0$
  $sync \leftarrow 1$
  return $\mathsf{Init}_{\mathsf{\$\text{-}sfAE}}()$

proc $\mathsf{Enc}_{\mathsf{\$\text{-}sfCCA}}(m)$
  $i \leftarrow i + 1$
  $c_i \leftarrow\$ \{0,1\}^{l(|m|)}$
  $c'_i \leftarrow \mathsf{Enc}_{\mathsf{\$\text{-}sfAE}}(m)$
  return $c_i$

proc $\mathsf{Dec}_{\mathsf{\$\text{-}sfCCA}}(c)$
  $j \leftarrow j + 1$
  if $j > i$ or $c \neq c_j$
    $sync \leftarrow 0$
  if $sync = 1$
    $\mathsf{Dec}_{\mathsf{\$\text{-}sfAE}}(c'_j)$
    $m \leftarrow \perp$
  else
    $m \leftarrow \mathsf{Dec}_{\mathsf{\$\text{-}sfAE}}(c)$
  return $m$

proc $\mathsf{Fin}_{\mathsf{\$\text{-}(sf)CCA}}(d)$
  return $\mathsf{Fin}_{\mathsf{\$\text{-}(sf)AE}}(1-d)$
```

Figure 4.4: The adversaries constructed in the proof of Theorem 4.9

These adversaries simulate the respective FL^0-(sf)AE game with the same bit b as in the FL-(sf)AE and therefore have the same advantage, so we have

$$\mathsf{Adv}^{\mathsf{FL}^0\text{-(sf)AE}}_{\mathcal{SE},\mathcal{A}_{\mathsf{FL}\text{-(sf)CCA}}}(k) \leq \mathsf{Adv}^{\mathsf{FL}\text{-(sf)AE}}_{\mathcal{SE},\mathcal{A}_{\mathsf{FL}\text{-(sf)AE}}}(k),$$

proving the statement. □

This theorem combined with Theorem 4.6 shows that the AE notions are not implied by integrity. Also we see below in Theorem 4.13 that AE notions imply integrity. This together with Theorem 4.5 and Theorem 4.3 shows that \$-sfCCA $\not\Rightarrow$ FtG-AE, so AE notions are indeed stronger than CCA notions.

The next two theorems are the equivalent of Theorem 3.2 and Theorem 3.3 and show that the sfAE notions are stronger than the AE notions. Their proofs can be applied directly and are therefore omitted.

Theorem 4.10 (FL-sfAE ⇒ FL-AE)
Let $\mathcal{SE}$ be a symmetric encryption scheme. There exists a constant c so that for any k, t, q and μ we have

$$\mathsf{Sec}^{\mathsf{FL\text{-}AE}}_{\mathcal{SE}}(k,t,q,\mu) \leq \mathsf{Sec}^{\mathsf{FL\text{-}sfAE}}_{\mathcal{SE}}(k,t+cq,q,\mu).$$

Theorem 4.11
Let $\mathcal{SE}$ be a stateless encryption scheme and take $\mathsf{FL} \in \{\mathsf{SEM}, \mathsf{FtG}, \mathsf{LoR}, \mathsf{RoR}, \$\}$ *. Then* $\mathsf{Sec}^{\mathsf{FL\text{-}sfAE}}_{\mathcal{SE}}(k,t,3,\mu) = 1$.

The same argument used here also can be used to show that \$-AE $\not\Rightarrow$ FtG-sfCCA, so the sfCCA and AE notions are incomparable.

4.3 Relation between AE-security and integrity

The following results are based on Theorem 1 and 2 from [Shr04], which consider the stateless scenario and the flavor RoR.

To prove the relations between AE-security and integrity we use two other games to bound advantages. The first we call FL^1-(sf)AE. It has the same queries as FL-(sf)AE, however the encryption and challenge queries always return the values of the game FL-(sf)$\mathsf{AE}_{\mathcal{SE}}(k,1)$, even if actually $b = 0$. The initialisation, finalisation and decryption queries work normally.

The second, $\mathsf{FL}^{\perp}$-(sf)AE, also has the same queries as FL-(sf)AE. Here the decryption queries always return the value of FL-(sf)AE with $b = 0$ (so always $\perp$). The initialisation, finalisation, encryption and challenge queries work normally.

Note that for an adversary $\mathcal{A}$ we have

$$\Pr[\mathsf{FL}\text{-(sf)}\mathsf{AE}^{\mathcal{A}}_{\mathcal{SE}}(k,1) = 1] = \Pr[\mathsf{FL}^1\text{-(sf)}\mathsf{AE}^{\mathcal{A}}_{\mathcal{SE}}(k,1) = 1],$$

$$\Pr[\mathsf{FL}^1\text{-}(\mathsf{sf})\mathsf{AE}^{\mathcal{A}}_{\mathcal{SE}}(k,0)=1]=\Pr[\mathsf{FL}^{\perp}\text{-}(\mathsf{sf})\mathsf{AE}^{\mathcal{A}}_{\mathcal{SE}}(k,1)=1]$$

and

$$\Pr[\mathsf{FL}\text{-}\mathsf{AE}^{\mathcal{A}}_{\mathcal{SE}}(k,0)=1]=\Pr[\mathsf{FL}^{\perp}\text{-}(\mathsf{sf})\mathsf{AE}^{\mathcal{A}}_{\mathcal{SE}}(k,0)=1]$$

Theorem 4.12 (FL-CPA + INT-(sf)CTXT ⇒ FL-(sf)AE)
Let $\mathsf{FL}\in\{\mathsf{SEM},\mathsf{FtG},\mathsf{LoR},\mathsf{RoR},\$\}$ *and let* $\mathcal{SE}$ *be a symmetric encryption scheme. For any* k, t, q *and* μ *we have*

$$\mathsf{Sec}^{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}_{\mathcal{SE}}(k,t,q,\mu)\le\mathsf{Sec}^{\mathsf{FL}\text{-}\mathsf{CPA}}_{\mathcal{SE}}(k,t,q,\mu)+\mathsf{Sec}^{\mathsf{INT}\text{-}(\mathsf{sf})\mathsf{CTXT}}_{\mathcal{SE}}(k,t,q,\mu).$$

Proof Let $\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}\in\mathfrak{A}^{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}_{(t,q,\mu)}$ be an adversary. We have

$$\begin{aligned}
&\mathsf{Adv}^{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}_{\mathcal{SE},\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}}(k)\\
&=\Pr[\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}^{\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}}_{\mathcal{SE}}(k,1)=1]-\Pr[\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}^{\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}}_{\mathcal{SE}}(k,0)=1]\\
&=\Pr[\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}^{\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}}_{\mathcal{SE}}(k,1)=1]-\Pr[\mathsf{FL}^1\text{-}(\mathsf{sf})\mathsf{AE}^{\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}}_{\mathcal{SE}}(k,0)=1]\\
&+\Pr[\mathsf{FL}^{\perp}\text{-}(\mathsf{sf})\mathsf{AE}^{\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}}_{\mathcal{SE}}(k,1)=1]-\Pr[\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}^{\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}}_{\mathcal{SE}}(k,0)=1]\\
&=\mathsf{Adv}^{\mathsf{FL}^1\text{-}(\mathsf{sf})\mathsf{AE}}_{\mathcal{SE},\mathcal{A}_{\mathsf{FL}\text{-}\mathsf{AE}}}(k)+\mathsf{Adv}^{\mathsf{FL}^{\perp}\text{-}(\mathsf{sf})\mathsf{AE}}_{\mathcal{SE},\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}}(k)
\end{aligned}$$

Note first that it is easy to construct an adversary $\mathcal{A}_{\mathsf{FL}\text{-}\mathsf{CPA}}$ with

$$\mathsf{Adv}^{\mathsf{FL}\text{-}\mathsf{CPA}}_{\mathcal{SE},\mathcal{A}_{\mathsf{FL}\text{-}\mathsf{CPA}}}(k)=\mathsf{Adv}^{\mathsf{FL}^{\perp}\text{-}(\mathsf{sf})\mathsf{AE}}_{\mathcal{SE},\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}}(k).$$

Simply run $\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}$ and respond to its decryption queries with $\perp$. This simulates the $\mathsf{FL}^{\perp}$-(sf)AE game with the same value of b.

Now consider the adversary $\mathcal{A}_{\mathsf{INT}\text{-}(\mathsf{sf})\mathsf{CTXT}}$ for the INT-(sf)CTXT game which runs $\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}$ (as an adversary for the FL^1-(sf)AE game) defined in Figure 4.5. This adversary uses the stated resources and simulates the FL^1-(sf)AE game up until the first time something other than $\perp$ is returned to $\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}$ for a decryption query. In stateless case this happens if $b=1$ and $\mathcal{A}_{\mathsf{FL}\text{-}\mathsf{AE}}$ queries a new ciphertext c so that $\mathsf{Dec}_K(c)\neq\perp$. In the stateful case this ciphertext must only be out-of-sync, not new. So consider the event W that $\mathcal{A}_{\mathsf{FL}\text{-}\mathsf{AE}}$ at some point queries a new (or out-of-sync) valid ciphertext. If this does not occur, then the values returned to $\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}$ in the entire game do not depend on b. So we have

$$\Pr[\mathsf{FL}^1\text{-}(\mathsf{sf})\mathsf{AE}^{\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}}_{\mathcal{SE}}(k,1)=1\mid\overline{W}]=\Pr[\mathsf{FL}^1\text{-}(\mathsf{sf})\mathsf{AE}^{\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}}_{\mathcal{SE}}(k,0)=1]$$

So

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{FL}^1\text{-}(\mathsf{sf})\mathsf{AE}}_{\mathcal{SE},\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}}(k)&=\Pr[W]\Pr[\mathsf{FL}^1\text{-}(\mathsf{sf})\mathsf{AE}^{\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}}_{\mathcal{SE}}(k,1)=1\mid W]\\
&+\Pr[\overline{W}]\Pr[\mathsf{FL}^1\text{-}(\mathsf{sf})\mathsf{AE}^{\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}}_{\mathcal{SE}}(k,0)=1]\\
&-\Pr[\mathsf{FL}^1\text{-}(\mathsf{sf})\mathsf{AE}^{\mathcal{A}_{\mathsf{FL}\text{-}(\mathsf{sf})\mathsf{AE}}}_{\mathcal{SE}}(k,0)=1]\\
&\le\Pr[W]
\end{aligned}$$

```
$\mathcal{A}_{\mathsf{INT\text{-}(sf)CTXT}}$
1 :  proc $\mathsf{Init}_{\mathsf{FL\text{-}(sf)AE}}()$
2 :    return $\mathsf{Init}_{\mathsf{INT\text{-}(sf)CTXT}}()$

3 :  proc $\mathsf{Enc}_{\mathsf{LoR\text{-}(sf)AE}}(m_0, m_1)$
4 :    $c \leftarrow \mathsf{Enc}_{\mathsf{INT\text{-}(sf)CTXT}}(m_1)$
5 :    return $c$

6 :  proc $\mathsf{Enc}_{\mathsf{RoR\text{-}(sf)AE}}(m_1)$
7 :    $c \leftarrow \mathsf{Enc}_{\mathsf{INT\text{-}(sf)CTXT}}(m_1)$
8 :    return $c$

9 :  proc $\mathsf{Enc}_{\mathsf{\$\text{-}(sf)AE}}(m)$
10 :   $c \leftarrow \mathsf{Enc}_{\mathsf{INT\text{-}(sf)CTXT}}(m)$
11 :   return $c$

12 : proc $\mathsf{Enc}_{\mathsf{FtG\text{-}(sf)AE}}(m)$
13 :   $c \leftarrow \mathsf{Enc}_{\mathsf{INT\text{-}(sf)CTXT}}(m)$
14 :   return $c$

18 : proc $\mathsf{Chal}_{\mathsf{FtG\text{-}(sf)AE}}(m_0, m_1)$
19 :   $c \leftarrow \mathsf{Enc}_{\mathsf{INT\text{-}(sf)CTXT}}(m_1)$
20 :   return $c$

21 : proc $\mathsf{Enc}_{\mathsf{SEM\text{-}(sf)AE}}(m)$
22 :   $c \leftarrow \mathsf{Enc}_{\mathsf{INT\text{-}(sf)CTXT}}(m)$
23 :   return $c$

24 : proc $\mathsf{Chal}_{\mathsf{SEM\text{-}(sf)AE}}(\tilde{\mathcal{M}})$
25 :   $m_1 \leftarrow\$ \tilde{\mathcal{M}}$
26 :   $c \leftarrow \mathsf{Enc}_{\mathsf{INT\text{-}(sf)CTXT}}(m_1)$
27 :   return $c$

28 : proc $\mathsf{Dec}_{\mathsf{FL\text{-}(sf)AE}}(c)$
29 :   $\mathsf{Dec}_{\mathsf{INT\text{-}(sf)CTXT}}(c)$
30 :   return $\perp$

31 : proc $\mathsf{Fin}_{\mathsf{FL\text{-}(sf)AE}}(\bar{d})$
32 :   return $\mathsf{Fin}_{\mathsf{INT\text{-}(sf)CTXT}}()$
```

Figure 4.5: The adversary used in the proof of Theorem 4.12.

But if W occurs, then $\mathcal{A}_{\mathsf{INT\text{-}(sf)CTXT}}$ wins its game, so

$$\mathsf{Adv}^{\mathsf{FL^1\text{-}(sf)AE}}_{\mathcal{SE},\mathcal{A}_{\mathsf{FL\text{-}(sf)AE}}}(k) \leq \mathsf{Adv}^{\mathsf{INT\text{-}(sf)CTXT}}_{\mathcal{SE},\mathcal{A}_{\mathsf{INT\text{-}(sf)CTXT}}}(k).$$

All together we have

$$\mathsf{Adv}^{\mathsf{FL\text{-}AE}}_{\mathcal{SE},\mathcal{A}_{\mathsf{FL\text{-}AE}}}(k) = \mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathcal{SE},\mathcal{A}_{\mathsf{INT\text{-}CTXT}}}(k) + \mathsf{Adv}^{\mathsf{FL\text{-}CPA}}_{\mathcal{SE},\mathcal{A}_{\mathsf{FL\text{-}CPA}}}(k),$$

proving the claim. □

Theorem 4.13 (FL-(sf)AE $\Rightarrow$ FL-CPA + INT-(sf)CTXT)
Let $\mathsf{FL} \in \{\mathsf{SEM}, \mathsf{FtG}, \mathsf{LoR}, \mathsf{RoR}, \$\}$ *and let* $\mathcal{SE}$ *be a symmetric encryption scheme. There exists a constant c so that for any k, t, q and μ we have*

$$\mathsf{Sec}^{\mathsf{FL\text{-}CPA}}_{\mathcal{SE}}(k, t, q, \mu) \leq \mathsf{Sec}^{\mathsf{FL\text{-}(sf)AE}}_{\mathcal{SE}}(k, t, q, \mu)$$

$$\mathsf{Sec}^{\mathsf{INT\text{-}(sf)CTXT}}_{\mathcal{SE}}(k, t, q, \mu) \leq 2 \cdot \mathsf{Sec}^{\mathsf{FL\text{-}(sf)AE}}_{\mathcal{SE}}(k, t + cq, q, \mu).$$

Proof For an FL-CPA adversary we can construct an FL-(sf)AE adversary with equal advantage by simply forwarding its queries. To prove the second statement let $\mathcal{A}_{\mathsf{INT\text{-}(sf)CTXT}}$ be an adversary for the INT-(sf)CTXT game. We begin by constructing an adversary $\mathcal{A}_{\mathsf{FL^1\text{-}(sf)AE}}$ for the FL^1-(sf)AE game.

```
$\mathcal{A}_{\mathsf{FL^1\text{-}(sf)AE}}$

1 :  proc $\mathsf{Init}_{\mathsf{INT\text{-}CTXT}}()$
2 :    $d \leftarrow 0$
3 :    return $\mathsf{Init}_{\mathcal{A}_{\mathsf{FL^1\text{-}(sf)AE}}}()$

4 :  proc $\mathsf{Enc}_{\mathsf{INT\text{-}CTXT}}(m)$
5 :    $c \leftarrow \mathsf{Enc}_{\mathsf{FL^1\text{-}(sf)AE}}(m)$
6 :    return $c$

7 :  proc $\mathsf{Dec}_{\mathsf{INT\text{-}CTXT}}(c)$
8 :    $m \leftarrow \mathsf{Dec}_{\mathsf{FL^1\text{-}(sf)AE}}(c)$
9 :    if $m \neq \perp$
10 :     $d \leftarrow 1$
11 :   return $m \neq \perp$

12 : proc $\mathsf{Fin}_{\mathsf{INT\text{-}CTXT}}()$
13 :   return $\mathsf{Fin}_{\mathsf{FL^1\text{-}(sf)AE}}(d)$
```

This adversary uses the stated resources. Note that if $b = 0$ in the $\mathsf{FL^1\text{-}(sf)AE}$ game then d is always 0, so $\Pr[\mathsf{FL^1\text{-}(sf)AE}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{FL^1\text{-}(sf)AE}}}(k,0) = 1] = 0$. For $b = 1$, $\mathcal{A}_{\mathsf{FL^1\text{-}(sf)AE}}$ simulates the $\mathsf{INT\text{-}CTXT}$ game and if $\mathcal{A}_{\mathsf{INT\text{-}(sf)CTXT}}$ wins the simulated game then d is set to 1.

$$\mathsf{Adv}_{\mathcal{SE},\mathcal{A}_{\mathsf{INT\text{-}(sf)CTXT}}}^{\mathsf{INT\text{-}(sf)CTXT}}(k) = \mathsf{Adv}_{\mathcal{SE},\mathcal{A}_{\mathsf{FL^1\text{-}(sf)AE}}}^{\mathsf{FL^1\text{-}(sf)AE}}(k)$$

Now note that

$$\begin{aligned}
&\mathsf{Adv}_{\mathcal{SE},\mathcal{A}_{\mathsf{FL^1\text{-}(sf)AE}}}^{\mathsf{FL^1\text{-}(sf)AE}}(k)\\
&= \Pr[\mathsf{FL^1\text{-}(sf)AE}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{FL^1\text{-}(sf)AE}}}(k,1) = 1] - \Pr[\mathsf{FL\text{-}(sf)AE}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{FL^1\text{-}(sf)AE}}}(k,0) = 1]\\
&+ \Pr[\mathsf{FL\text{-}(sf)AE}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{FL^1\text{-}(sf)AE}}}(k,0) = 1] - \Pr[\mathsf{FL^1\text{-}(sf)AE}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{FL\text{-}(sf)AE}}}(k,0) = 1]\\
&= \mathsf{Adv}_{\mathcal{SE},\mathcal{A}_{\mathsf{FL^1\text{-}(sf)AE}}}^{\mathsf{FL\text{-}(sf)AE}}(k) + \Pr[\mathsf{FL^\perp\text{-}(sf)AE}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{FL^1\text{-}(sf)AE}}}(k,0) = 1]\\
&- \Pr[\mathsf{FL^\perp\text{-}(sf)AE}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{FL\text{-}(sf)AE}}}(k,1) = 1]
\end{aligned}$$

Now consider an $\mathsf{FL\text{-}(sf)AE}$ adversary $\mathcal{A}$ which runs $\mathcal{A}_{\mathsf{FL^1\text{-}(sf)AE}}$ and forwards its queries except for the decryption queries, which are answered with $\perp$. The bit in the finalisation query is flipped before being forwarded. This simulates the $\mathcal{A}_{\mathsf{FL^1\text{-}(sf)AE}}$ game with the same value of b, so

$$\begin{aligned}
&\Pr[\mathsf{FL^\perp\text{-}(sf)AE}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{FL^1\text{-}(sf)AE}}}(k,0) = 1] - \Pr[\mathsf{FL^\perp\text{-}(sf)AE}_{\mathcal{SE}}^{\mathcal{A}_{\mathsf{FL\text{-}(sf)AE}}}(k,1) = 1]\\
&\leq \mathsf{Adv}_{\mathcal{SE},\mathcal{A}}^{\mathsf{FL\text{-}(sf)AE}}(k)
\end{aligned}$$

This concludes the proof. □

We finish this chapter by giving an overview of the notions and relation so far. Figure 4.6 depicts all of the notions for a single flavor along with the

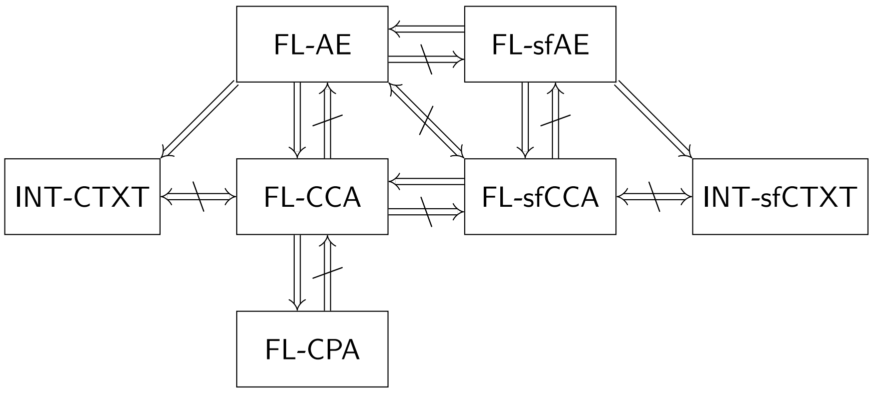

Figure 4.6: An overview of the relations between different notions for a flavor FL $\in$ $\{$SEM, FtG, LoR, RoR, \$$\}$.

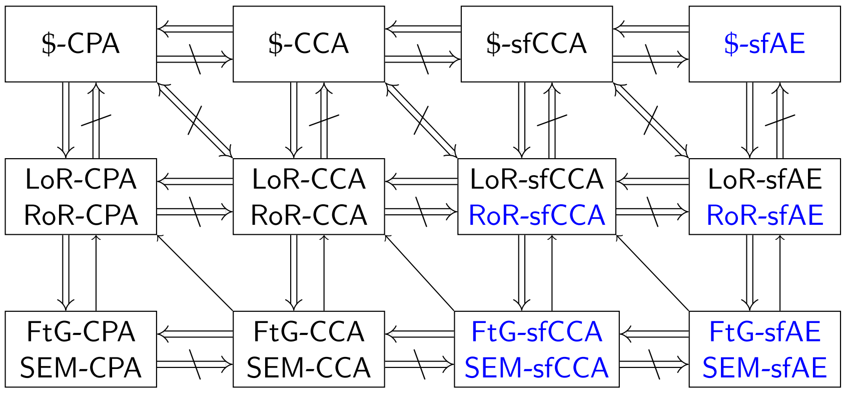

Figure 4.7: An overview over most of the notions defined so far. Notions in the same field are equivalent, notions defined for the first time are in blue.

ciphertext integrity notions. Figure 4.7 depicts an overview over most of the notions defined so far. The notions of integrity and stateless authenticated encryption are omitted for readability. The sfCCA notions could be replaced by the corresponding AE notions without changing any arrows.

Chapter 5

Progress-hiding encryption schemes

Consider an adversary which does not have access to a channel for the entire communication. For example, the adversary could miss the beginning of communication. It could also have access at the beginning, then not observe the channel for while and then return. We wish to hide from such an adversary the progress of the channel, so it should not be able to tell whether messages were sent in its absence. We call this goal ”progress-hiding”.

The most obvious way that this information can be leaked is through sequence numbers which are used as nonces and sent along with the ciphertext. This issue has been recognized, for example in [BNT19], where the authors construct schemes which ensure that nonces are not visible. There are however other ways progress can be leaked. In particular, an adversary can abuse the correctness and integrity of a channel to learn things about its progress. To properly analyse these attacks, we now formalize notions of progress-hiding for active and passive adversaries, and show the relations between these notions and the other notions defined so far.

5.1 Progress-hiding chosen-plaintext security

We begin with a passive adversary. We wish to ensure that ciphertexts do not leak the number of messages that have been encrypted so far. We in fact introduce two notions which provide progress-hiding encryption. Both follow a similar idea. The adversary has access to a progress oracle, which takes a plaintext and encrypts it if $b = 1$. This oracle represents messages which might be sent when the adversary is absent. The adversary also has access to a truthful encryption oracle, which allows it to examine ciphertexts. This oracle represents the messages sent when the adversary is observing the channel.

In the first notion, prePH-CPA, the adversary has access to these oracles in

```
Game prePH-CPA$_{\mathcal{SE}}(k,b)$

proc Init$_{\text{prePH-CPA}}()$
    $K \leftarrow\$ \mathsf{KGen}(k)$
    $phase \leftarrow 0$
    return $k$

proc Prog$_{\text{prePH-CPA}}(m)$
    if $b = 1$ and $phase = 0$
        $\mathsf{Enc}_K(m)$
    return $\perp$

proc Enc$_{\text{prePH-CPA}}(m)$
    $phase \leftarrow 1$
    $c \leftarrow \mathsf{Enc}_K(m)$
    return $c$

proc Fin$_{\text{prePH-CPA}}(d)$
    return $d$
```

```
Game PH-CPA$_{\mathcal{SE}}(k,b)$

proc Init$_{\text{PH-CPA}}()$
    $K \leftarrow\$ \mathsf{KGen}(k)$

    return $k$

proc Prog$_{\text{PH-CPA}}(m)$
    if $b = 1$
        $\mathsf{Enc}_K(m)$
    return $\perp$

proc Enc$_{\text{PH-CPA}}(m)$

    $c \leftarrow \mathsf{Enc}_K(m)$
    return $c$

proc Fin$_{\text{PH-CPA}}(d)$
    return $d$
```

Figure 5.1: The games for prePH-CPA and PH-CPA.

two phases, first only to the progress oracle and then only to the encryption oracle. This models an attack where the adversary misses the beginning of the communication over a channel. In the second notion, PH-CPA, the adversary has access to both oracles the entire game. This therefore additionally covers scenarios where the adversary is absent in the middle of the communication.

Definition 5.1 ((pre)PH-CPA)
The progress-hiding chosen-plaintext security of an encryption scheme $\mathcal{SE}$ with security parameter k against adversaries with runtime t using q queries totalling μ bits is defined by $\mathsf{Sec}^{\mathsf{(pre)PH\text{-}CPA}}_{\mathcal{SE}}(k,t,q,\mu)$ *using the game* (pre)PH-CPA$_{\mathcal{SE}}(k,b)$ *depicted in Figure 5.1. The input for the progress and encryption queries are messages $m \in \mathcal{M}$, the input for the finalization query is a bit d.*

Clearly PH-CPA implies prePH-CPA, as we can forward all the the queries of an adversary for prePH-CPA in a PH-CPA game. The following theorem shows that PH-CPA is in fact stronger than prePH-CPA. This shows that it may easier for adversary to learn something about the progress of a channel if it saw some messages before it was absent.

Theorem 5.2
Let $\mathcal{SE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an encryption scheme and fix q. Then there exists*

an encryption scheme $\mathcal{SE}'$ and a constant c so that for any k, t and μ we have

$$\begin{aligned}\mathsf{Sec}^{\mathsf{prePH\text{-}CPA}}_{\mathcal{SE}'}(k,t,q,\mu) &= \mathsf{Sec}^{\mathsf{prePH\text{-}CPA}}_{\mathcal{SE}}(k,t+cq,q,\mu)\\ \mathsf{Sec}^{\mathsf{PH\text{-}CPA}}_{\mathcal{SE}'}(k,t,3,\mu) &= 1\end{aligned}$$

Proof Let l be the length of q in a binary representation. Consider the following scheme $\mathcal{SE}'$ which adds a counter i to each ciphertext, but XORs the counter with a random string beforehand:

Scheme $\mathcal{SE}'$

```
proc $\mathsf{KGen}_{\mathcal{SE}'}(k)$
    $K \leftarrow\$ \mathsf{KGen}_{\mathcal{SE}}(k)$
    $i \leftarrow 0$
    $\kappa \leftarrow \{0,1\}^l$
    return $(K,(i,\kappa))$

proc $\mathsf{Enc}_{\mathcal{SE}'}(K,m,(i,\kappa))$
    $i \leftarrow i+1$
    $c \leftarrow (\kappa \oplus i, \mathsf{Enc}_{\mathcal{SE}}(K,m))$
    return $(c,(i,\kappa))$

proc $\mathsf{Dec}_{\mathcal{SE}'}(K,c)$
    $(\kappa',c') \leftarrow c$
    $m \leftarrow \mathsf{Dec}_{\mathcal{SE}}(K,c')$
    return $m$
```

The counter i is encoded here as a bitstring of length l, so that the $\kappa \oplus i$ is well-defined. We can construct an adversary for the prePH-CPA game against $\mathcal{SE}$ that runs an adversary against $\mathcal{SE}'$. In the first phase the new adversary forwards the progress queries. In the second it samples κ itself, sets i to 1 and updates accordingly. As κ is sampled randomly, the distribution of the returned ciphertexts is the same independently of the value of b in the game against $\mathcal{SE}$. So we simulate the game against $\mathcal{SE}'$, proving the first statement.

For the second consider an adversary for the PH-CPA game against $\mathcal{SE}'$ which sends an encryption query, then a progress query and finally another encryption query. By XORing the prefixes of the result of the encryption queries the adversary removes κ, so the result it either $1 \oplus 2$ or $1 \oplus 3$ depending on the value of b, so the adversary can achieve the claimed advantage.□

The scheme SE' has a real-world inspiration: TLS 1.3 uses a counter XORed with a key as a nonce input for its encryption algorithm. However, as this nonce is not sent as part of the ciphertext, TLS 1.3 can achieve progress-hiding. We will analyse this in more detail in Chapter 6.

Having established that there is a reason to consider PH-CPA instead of prePH-CPA, we only use the stronger notion in the following. Let us now examine the relations of PH-CPA with the notions defined so far.

Theorem 5.3 (LoR-sfAE $\not\Rightarrow$ PH-CPA)
Let $\mathcal{SE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme. Then there exists an encryption scheme $\mathcal{SE}'$ and a constant c so that for any k, t, q and μ we have

$$\mathsf{Sec}^{\mathsf{LoR\text{-}sfAE}}_{\mathcal{SE}'}(k,t,q,\mu) = \mathsf{Sec}^{\mathsf{LoR\text{-}sfCCA}}_{\mathcal{SE}}(k,t+cq,q,\mu)$$
$$\mathsf{Sec}^{\mathsf{PH\text{-}CPA}}_{\mathcal{SE}'}(k,t,q,\mu) = 1$$

Proof For a scheme $\mathcal{SE}$ consider the following scheme $\mathcal{SE}'$, which adds a counter to the message before encrypting using $\mathsf{Enc}_{\mathcal{SE}}$ and also adds the counter to the resulting ciphertext.

Scheme $\mathcal{SE}'$

```
1 : proc $\mathsf{KGen}_{\mathcal{SE}'}(k)$
2 :   $K \leftarrow\$ \mathsf{KGen}_{\mathcal{SE}}(k)$
3 :   $i \leftarrow 0$
4 :   return $(K, i)$

5 : proc $\mathsf{Enc}_{\mathcal{SE}'}(K, m, i)$
6 :   $i \leftarrow i + 1$
7 :   $c \leftarrow (i, \mathsf{Enc}_{\mathcal{SE}}(K, i \| m))$
8 :   return $(c, i)$

12 : proc $\mathsf{Dec}_{\mathcal{SE}'}(K, c)$
13 :   $(i, c') \leftarrow c$
14 :   $i' \| m \leftarrow \mathsf{Dec}_{\mathcal{SE}}(K, c')$
15 :   if $i \neq i'$
16 :     $m \leftarrow \perp$
17 :   return $m$
```

For an adversary $\mathcal{A}_{\mathsf{LoR\text{-}sfAE}}$ against $\mathcal{SE}'$ in the LoR-sfAE game we can construct an adversary against $\mathcal{SE}$ that keeps track of i. It can then add i to every messages and ciphertext in encryption queries and remove it from every ciphertext and message in decryption queries (with a check to ensure that they have the same value). This simulates the game against $\mathcal{SE}'$ with the same value b as in the actual game against $\mathcal{SE}$.

Now consider an adversary for the PH-CPA game against $\mathcal{SE}'$ which sends first a progress query and then an encryption query. Depending on the value of b, the counter will either be 1 or 2, allowing the adversary to achieve advantage 1. □

So LoR-sfAE does not ensure progress-hiding. We now see that \$-CPA does. This means that ensuring progress-hiding is a reason why channels should try to achieve privacy using \$-type encryption oracle instead of LoR ones.

Theorem 5.4 (\$-CPA $\Rightarrow$ PH-CPA)
Let $\mathcal{SE}$ be a symmetric encryption scheme. There exists a constant c so that for any k, t, q and μ we have

$$\mathsf{Sec}^{\mathsf{PH\text{-}CPA}}_{\mathcal{SE}}(k,t,q,\mu) \leq 2 \cdot \mathsf{Sec}^{\mathsf{\$\text{-}CPA}}_{\mathcal{SE}}(k,t+cq,q,\mu).$$

Proof Consider the following adversary

```
$\mathcal{A}_{\text{\$-CPA}}$

proc $\mathsf{Init}_{\text{PH-CPA}}()$
  $b' \leftarrow\$ \{0,1\}$
  return $\mathsf{Init}_{\text{\$-CPA}}()$

proc $\mathsf{Prog}_{\text{PH-CPA}}(m)$
  if $b' = 1$
    $\mathsf{Enc}_{\text{\$-CPA}}(m)$
  return $\perp$

proc $\mathsf{Enc}_{\text{PH-CPA}}(m)$
  $c \leftarrow \mathsf{Enc}_{\text{\$-CPA}}(m)$
  return $c$

proc $\mathsf{Fin}_{\text{PH-CPA}}(d)$
  return $\mathsf{Fin}_{\text{\$-CPA}}(d = b')$
```

In the game $\text{\$-CPA}_{\mathcal{SE}}(k,0)$, the ciphertexts returned for $\mathsf{Enc}_{\text{PH-CPA}}$ queries are sampled randomly and are therefore independent of b'. In the game $\text{\$-CPA}_{\mathcal{SE}}(k,1)$, $\mathcal{A}_{\text{\$-CPA}}$ simulates the game $\text{PH-CPA}_{\mathcal{SE}}(k,b')$. Using the same argument as in Theorem 2.6 gives the stated result. □

5.2 Progress-hiding chosen-ciphertext security

We now turn to active adversaries. To model this we give the adversary a decryption oracle. In addition, the progress query not only encrypts messages if $b = 1$, but also decrypts the resulting ciphertext. We present three notions which formalize this idea. The first naive attempt PH-CCA makes only these changes.

Definition 5.5 (PH-CCA)
The progress-hiding chosen-ciphertext security of an encryption scheme $\mathcal{SE}$ with security parameter k against adversaries with runtime t using q queries totalling μ bits is defined by $\mathsf{Sec}^{\text{PH-CCA}}_{\mathcal{SE}}(k,t,q,\mu)$ using the game $\text{PH-CCA}_{\mathcal{SE}}(k,b)$ depicted on the left-hand side of Figure 5.2. The input for the progress and encryption queries are messages $m \in \mathcal{M}$, the input for the decryption queries are strings c and the input for the finalization query is a bit d.

The next theorem shows that this notion can not be achieved by schemes which fulfil the stateful correctness and ciphertext integrity notions. As progress-hiding is a property desired for channels, this notion is therefore not very useful.

Theorem 5.6
Let $\mathcal{SE}$ be a scheme which fulfils the stateful correctness and ciphertext integrity notions. Then there exist t and μ so that for every k we have $\mathsf{Sec}^{\text{PH-CCA}}_{\mathcal{SE}}(k,t,3,\mu) = 1$.

Proof Consider an adversary which sends an encryption query, then sends a progress query and finally sends a decryption query with the ciphertext

```
Game PH-CCA$_{\mathcal{SE}}(k, b)$

1 :  proc Init$_{\text{PH-CCA}}()$
2 :    $K \leftarrow\$ \mathsf{KGen}(k)$
3 :    return $k$

4 :  proc Prog$_{\text{PH-CCA}}(m)$
5 :    if $b = 1$
6 :      $c \leftarrow \mathsf{Enc}_K(m)$
7 :      $\mathsf{Dec}_K(c)$
8 :    return $\perp$

9 :  proc Enc$_{\text{PH-CCA}}(m)$
10 :   $c \leftarrow \mathsf{Enc}_K(m)$
11 :   return $c$

12 : proc Dec$_{\text{PH-CCA}}(c)$
13 :   $m \leftarrow \mathsf{Dec}_K(c)$
14 :   return $m$

15 : proc Fin$_{\text{PH-CCA}}(d)$
16 :   return $d$
```

```
Game PH-sfCCA$_{\mathcal{SE}}(k, b)$

1 :  proc Init$_{\text{PH-sfCCA}}()$
2 :    $K \leftarrow\$ \mathsf{KGen}(k)$
3 :    $i \leftarrow 0$
4 :    $j \leftarrow 0$
5 :    $sync \leftarrow 1$
6 :    return $k$

7 :  proc Prog$_{\text{PH-sfCCA}}(m)$
8 :    if $b = 1$ and $(sync = 0$ or $i = j)$
9 :      $c \leftarrow \mathsf{Enc}_K(m)$
10 :     $\mathsf{Dec}_K(c)$
11 :   return $\perp$

12 : proc Enc$_{\text{PH-sfCCA}}(m)$
13 :   $i \leftarrow i + 1$
14 :   $c_i \leftarrow \mathsf{Enc}_K(m)$
15 :   return $c$

16 : proc Dec$_{\text{PH-sfCCA}}(c)$
17 :   $j \leftarrow j + 1$
18 :   $m \leftarrow \mathsf{Dec}_K(c)$
19 :   if $j > i$ or $c \neq c_j$
20 :     $sync \leftarrow 0$
21 :   return $m$

22 : proc Fin$_{\text{PH-sfCCA}}(d)$
23 :   return $d$
```

Figure 5.2: The games for the notions PH-CCA and PH-sfCCA. The notion prePH-sfCCA is identical to PH-sfCCA except replacing line 8 with "**if** $b = 1$ and $sync = 1$ and $i = j$".

returned from the encryption query. Depending on the value of b, this query is either in-sync or out-of-sync and by assumption the scheme returns something different for each of these options, allowing the adversary to achieve advantage 1. $\square$

The problem with the notion above is that the adversary can change whether the scheme is in-sync with a progress query. So to define a meaningful notion of progress-hiding for channels, we need to ensure that progress queries cannot alter sync. Adding this restriction does not contradict the attacks we are trying to protect against: The progress queries model messages sent while the adversary is absent. As these messages represent normal communication, they should never break sync. Formally, we can do this by requiring that $i = j$ for progress queries while the game is still in-sync.

We actually present two notions that capture this idea. In the first notion prePH-sfCCA, we assume that the adversary does not tamper with the messages before being absent. Formally, this means we only allow progress queries while the game is in-sync. In the second notion, PH-sfCCA, we allow out-of-sync progress queries. Clearly PH-sfCCA implies prePH-sfCCA. We see below that PH-sfCCA is indeed stronger, meaning that a scheme could (in theory) leak more progress information if the adversary is allowed to tamper with the messages before it is absent. This is of course mostly a theoretical difference, as for actual schemes, having integrity should already cause the channel to close if tampering occurs.

Definition 5.7 ((pre)PH-sfCCA)
The stateful progress-hiding chosen-ciphertext security of an encryption scheme $\mathcal{SE}$ with security parameter k against adversaries with runtime t using q queries totalling μ bits is defined by $\mathsf{Sec}^{\mathsf{PH\text{-}sfCCA}}_{\mathcal{SE}}(k,t,q,\mu)$ using the game $\mathsf{PH\text{-}sfCCA}_{\mathcal{SE}}(k,b)$ depicted on the right-hand side of Figure 5.2. The input for the progress and encryption queries are messages $m \in \mathcal{M}$, the input for the decryption queries are strings c and the input for the finalization query is a bit d.

Similarly we define $\mathsf{Sec}^{\mathsf{prePH\text{-}sfCCA}}_{\mathcal{SE}}(k,t,q,\mu)$ using the game $\mathsf{prePH\text{-}sfCCA}_{\mathcal{SE}}(k,b)$ also depicted on the right-hand side of Figure 5.2.

We begin our analysis with some trivial implications: we have

$$\mathsf{PH\text{-}CCA} \Rightarrow \mathsf{PH\text{-}sfCCA} \Rightarrow \mathsf{prePH\text{-}sfCCA} \Rightarrow \mathsf{PH\text{-}CPA}$$

and $\mathsf{PH\text{-}sfCCA} \not\Rightarrow \mathsf{PH\text{-}CCA}$.

The separation $\mathsf{prePH\text{-}sfCCA} \not\Rightarrow \mathsf{PH\text{-}sfCCA}$ is the subject of the next theorem. The separation $\mathsf{PH\text{-}CPA} \not\Rightarrow \mathsf{prePH\text{-}sfCCA}$ is a corollary of Theorem 5.9 below.

Theorem 5.8 (prePH-sfCCA $\not\Rightarrow$ PH-sfCCA)
Let $\mathcal{SE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme which fulfils $\mathsf{INT\text{-}sfCTXT}$. Then there exists an encryption scheme $\mathcal{SE}'$ and a constant c so that for any k, t, q

and μ we have

$$\begin{aligned}\mathsf{Sec}^{\mathsf{prePH\text{-}sfCCA}}_{\mathcal{SE}'}(k,t,q,\mu) &= \mathsf{Sec}^{\mathsf{prePH\text{-}sfCCA}}_{\mathcal{SE}}(k,t+cq,q,\mu)\\ \mathsf{Sec}^{\mathsf{PH\text{-}sfCCA}}_{\mathcal{SE}'}(k,t,q,\mu) &= 1\end{aligned}$$

We could also prove this theorem with a weaker assumption on $\mathcal{SE}$. Instead of requiring INT-sfCTXT, it would suffice to require that the output of the first out-of-sync call to the decryption algorithm of $\mathcal{SE}$ is somehow recognizable to $\mathcal{SE}'$. This is somewhat tedious to formalize, so we use the stronger assumption instead.

Proof Consider the scheme $\mathcal{SE}'$ below which counts the number of out-of-sync queries and prepends the counter to the output of each out-of-sync decryption query.

Scheme $\mathcal{SE}'$

```
1 : proc KGen_{SE'}(k)
2 :   $i \leftarrow 0$
3 :   $sync \leftarrow 1$
4 :   $K \leftarrow \mathsf{KGen}_{\mathcal{SE}}(k)$
5 :   return $(K, (i, sync))$

6 : proc Enc_{SE'}(K, m)
7 :   $c \leftarrow \mathsf{Enc}_{\mathcal{SE}}(K, m)$
8 :   return $c$

9 : proc Dec_{SE'}(K, c, (i, sync))
10 :   $m \leftarrow \mathsf{Dec}_{\mathcal{SE}}(K, c)$
11 :   if $m = \bot$
12 :     $sync \leftarrow 0$
13 :   if $sync = 0$
14 :     $i \leftarrow i + 1$
15 :     $m \leftarrow i \| m$
16 :   return $(m, (i, sync))$
```

For an adversary against $\mathcal{SE}'$ in the prePH-sfCCA we can construct an adversary against $\mathcal{SE}$ which forwards all queries and keeps track of sync. If sync is broken, then it initialises a counter and adds it when it returns ciphertexts. As the adversary cannot send progress queries after sync is broken, this simulates the game against $\mathcal{SE}'$ with the same value of b.

Now consider an adversary $\mathcal{A}$ for the PH-sfCCA game which sends a decryption query, then a progress query and then another decryption query. As all decryption queries are out-of-sync, the counter will either be equal to 2 or 3, depending on the value of b. This allows an adversary to achieve advantage 1. □

Now let us examine whether any of the notions defined so far imply this stronger notion of progress-hiding. To achieve PH-CPA it was enough to assume \$-CPA. As the next theorem shows, privacy is not enough to provide progress-hiding against an active adversary.

Theorem 5.9 (\$-sfCCA $\not\Rightarrow$ prePH-sfCCA)
Let $\mathcal{SE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme which fulfils INT-sfCTXT. *Then there exists an encryption scheme $\mathcal{SE}'$ and a constant c so that for any k, t, q and μ we have*

$$\mathsf{Sec}^{\$\text{-sfCCA}}_{\mathcal{SE}'}(k,t,q,\mu) = \mathsf{Sec}^{\$\text{-sfCCA}}_{\mathcal{SE}}(k,t+cq,q,\mu)$$
$$\mathsf{Sec}^{\mathsf{prePH\text{-}sfCCA}}_{\mathcal{SE}'}(k,t,q,\mu) = 1$$

Again the assumption that $\mathcal{SE}$ fulfils INT-sfCTXT is stronger than necessary, but allows for an easier proof.

Proof Consider the following scheme $\mathcal{SE}'$, which adds the total number of decryption queries (including in-sync ones) to the output of out-of-sync decryption queries.

Scheme $\mathcal{SE}'$

```
1 :  proc $\mathsf{KGen}_{\mathcal{SE}'}(k)$
2 :    $i \leftarrow 0$
3 :    $K \leftarrow \mathsf{KGen}_{\mathcal{SE}}(k)$
4 :    return $(K, i)$

5 :  proc $\mathsf{Enc}_{\mathcal{SE}'}(K, m)$
6 :    $c \leftarrow \mathsf{Enc}_{\mathcal{SE}}(K, m)$
7 :    return $c$

8 :  proc $\mathsf{Dec}_{\mathcal{SE}'}(K, c, i)$
9 :    $i \leftarrow i + 1$
10 :   $m \leftarrow \mathsf{Dec}_{\mathcal{SE}}(K, c)$
11 :   if $m = \perp$
12 :     $m \leftarrow i \| m$
13 :   return $(m, i)$
```

For an adversary against $\mathcal{SE}'$ in the \$-sfCCA game we can construct an adversary against $\mathcal{SE}$ which updates and adds i. As every $\mathsf{Dec}_{\$\text{-sfCCA}}$ query causes a call to the decryption algorithm, this simulates the game against $\mathcal{SE}$ with the same value of b.

On the other hand we can construct an adversary for the prePH-sfCCA game which sends one progress query and then an arbitrary decryption query. As this query is out-of-sync, the adversary learns if 1 or 2 calls to the decryption algorithm have been made and can therefore achieve advantage 1. □

The schemes constructed in the last two proofs give some additional intuition about what the notion PH-sfCCA protects against. Clearly in-sync decryption queries cannot leak progress information, as they must return the plaintext by correctness. Insecurity can arise when out-of-sync queries leak progress messages. Intuitively, assuming that a scheme has integrity should solve this problem, as schemes with integrity always return $\perp$ for out-of-sync queries. Indeed, the two schemes constructed above did not have integrity. The next theorem shows that integrity (along with PH-CPA) indeed implies PH-sfCCA, a sort of generic composition for progress-hiding.

Theorem 5.10 (PH-CPA + INT-sfCTXT ⇒ PH-sfCCA)
Let $\mathcal{SE}$ be a symmetric encryption scheme which fulfils stateful correctness. For any k, t, q and μ we have

$$\mathsf{Sec}^{\text{PH-sfCCA}}_{\mathcal{SE}}(k,t,q,\mu) \leq \mathsf{Sec}^{\text{PH-CPA}}_{\mathcal{SE}}(k,t+c_1q,q,\mu) + 2\cdot \mathsf{Sec}^{\text{INT-sfCTXT}}_{\mathcal{SE}}(k,t,q,\mu).$$

A direct corollary of this theorem is that \$-sfAE implies PH-sfCCA.

Before we start the proof we make a remark on the correctness assumption. For the privacy and integrity notions we have defined it does not matter whether the scheme is actually correct. Queries given by correctness are suppressed in the privacy games and do not matter in the integrity games. Here we do not suppresses these queries, so a scheme could leak progress-information by decrypting incorrectly. Ensuring correctness will be an important part of Chapter 7.

Proof Consider the following game:

```
Game  $\mathsf{G}_{\mathcal{SE}}(k,b,b')$

1 :  proc $\mathsf{Init}_\mathsf{G}()$
2 :     $K \leftarrow\!\$\ \mathsf{KGen}(k)$
3 :     $i \leftarrow 0$
4 :     $j \leftarrow 0$
5 :     $sync \leftarrow 1$
6 :     return $k$

7 :  proc $\mathsf{Prog}_\mathsf{G}(m)$
8 :     if $b = 1$ and $(sync = 0$ or $i = j)$
9 :        $c \leftarrow \mathsf{Enc}_K(m)$
10 :       $\mathsf{Dec}_K(c)$
11 :    return $\perp$

12 : proc $\mathsf{Enc}_\mathsf{G}(m)$
13 :    $i \leftarrow i + 1$
14 :    $c_i \leftarrow \mathsf{Enc}_K(m)$
15 :    return $c$

16 : proc $\mathsf{Dec}_\mathsf{G}(c)$
17 :    $j \leftarrow j + 1$
18 :    $m \leftarrow \mathsf{Dec}_K(c)$
19 :    if $j > i$ or $c \neq c_j$
20 :       $sync \leftarrow 0$
21 :    if $sync = 0$ and $b' = 0$
22 :       $m \leftarrow \perp$
23 :    return $m$

24 : proc $\mathsf{Fin}_\mathsf{G}(d)$
25 :    return $d$
```

This game is almost the same as the PH-sfCCA. The only difference is that for $b' = 0$ only the decryption of in-sync decryption query is returned, for out-of-sync queries the oracle returns $\perp$. So the adversary can learn nothing from decryption queries if $b' = 0$, as the answer to in-sync decryption queries is given by correctness. Now let $\mathcal{A}_{\text{PH-sfCCA}}$ be an adversary for the PH-sfCCA game. As the games are the same for $b' = 1$ we have

$$\mathsf{Adv}^{\text{PH-sfCCA}}_{\mathcal{SE},\mathcal{A}_{\text{PH-sfCCA}}}(k) = \Pr[\mathsf{G}^{\mathcal{A}_{\text{PH-sfCCA}}}_{\mathcal{SE}}(k,1,1) = 1] - \Pr[\mathsf{G}^{\mathcal{A}_{\text{PH-sfCCA}}}_{\mathcal{SE}}(k,0,1) = 1]$$

```
$\mathcal{A}_{\text{PH-CPA}}$

proc $\mathsf{Init}_{\text{PH-sfCCA}}()$
    $K \leftarrow\$ \mathsf{Init}_{\text{PH-CPA}}()$
    $i \leftarrow 0$
    $j \leftarrow 0$
    $sync \leftarrow 1$
    return $k$

proc $\mathsf{Prog}_{\text{PH-sfCCA}}(m)$
    if $sync = 0$ or $i = j$
        $\mathsf{Prog}_{\text{PH-CPA}}(m)$
    return $\perp$

proc $\mathsf{Enc}_{\text{PH-sfCCA}}(m)$
    $i \leftarrow i + 1$
    $m_i \leftarrow m$
    $c_i \leftarrow \mathsf{Enc}_{\text{PH-CPA}}(m)$
    return $c$

proc $\mathsf{Dec}_{\text{PH-sfCCA}}(c)$
    $j \leftarrow j + 1$
    if $j > i$ or $c \neq c_j$
        $sync \leftarrow 0$
    if $sync = 1$
        $m \leftarrow m_j$
    else
        $m \leftarrow \perp$
    return $m$

proc $\mathsf{Fin}_{\text{PH-sfCCA}}(d)$
    return $\mathsf{Fin}_{\text{PH-CPA}}(d)$
```

```
$\mathcal{A}_{\text{INT-sfCTXT}}$

proc $\mathsf{Init}_{\text{PH-sfCCA}}()$
    $k \leftarrow \mathsf{Init}_{\text{INT-sfCTXT}}()$
    $i \leftarrow 0$
    $j \leftarrow 0$
    $sync \leftarrow 1$
    return $k$

proc $\mathsf{Prog}_{\text{PH-sfCCA}}(m)$
    if $sync = 0$ or $i = j$
        $c \leftarrow \mathsf{Enc}_{\text{INT-sfCTXT}}(m)$
        $\mathsf{Dec}_{\text{INT-sfCTXT}}(c)$
    return $\perp$

proc $\mathsf{Enc}_{\text{PH-sfCCA}}(m)$
    $i \leftarrow i + 1$
    $c_i \leftarrow \mathsf{Enc}_{\text{INT-sfCTXT}}(m)$
    return $c_i$

proc $\mathsf{Dec}_{\text{PH-sfCCA}}(c)$
    $j \leftarrow j + 1$
    $\mathsf{Dec}_{\text{INT-sfCTXT}}(c)$
    if $j > i$ or $c \neq c_j$
        $sync \leftarrow 0$
    if $sync = 1$
        $m \leftarrow m_j$
    else
        $m \leftarrow \perp$
    return $m$

proc $\mathsf{Fin}_{\text{PH-sfCCA}}(d)$
    return $\mathsf{Fin}_{\text{INT-sfCTXT}}()$
```

Figure 5.3: The adversaries used in the proof of Theorem 5.10.

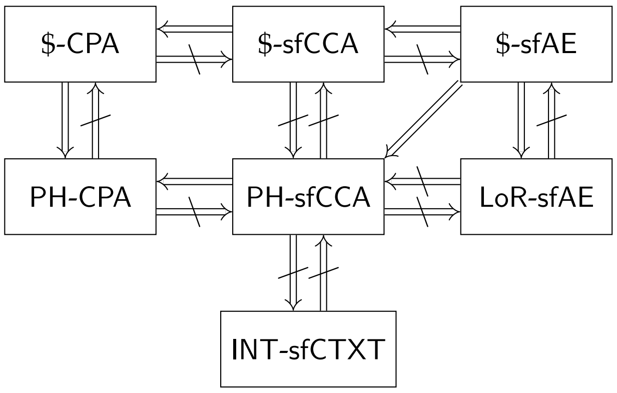

Figure 5.4: Relations between notions of progress-hiding.

Now we add and subtract the term $\Pr[\mathsf{G}_{\mathcal{SE}}^{\mathcal{A}_{\text{PH-sfCCA}}}(k,0,0) = 1]$ and the term $\Pr[\mathsf{G}_{\mathcal{SE}}^{\mathcal{A}_{\text{PH-sfCCA}}}(k,1,0) = 1]$ and bound the advantage of $\mathcal{A}_{\text{PH-sfCCA}}$ by the sum of its advantage in three different games:

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{SE},\mathcal{A}_{\text{PH-sfCCA}}}^{\mathsf{G}_1}(k) &= \Pr[\mathsf{G}_{\mathcal{SE}}^{\mathcal{A}_{\text{PH-sfCCA}}}(k,1,1) = 1] - \Pr[\mathsf{G}_{\mathcal{SE}}^{\mathcal{A}_{\text{PH-sfCCA}}}(k,1,0) = 1] \\
\mathsf{Adv}_{\mathcal{SE},\mathcal{A}_{\text{PH-sfCCA}}}^{\mathsf{G}_2}(k) &= \Pr[\mathsf{G}_{\mathcal{SE}}^{\mathcal{A}_{\text{PH-sfCCA}}}(k,1,0) = 1] - \Pr[\mathsf{G}_{\mathcal{SE}}^{\mathcal{A}_{\text{PH-sfCCA}}}(k,0,0) = 1] \\
\mathsf{Adv}_{\mathcal{SE},\mathcal{A}_{\text{PH-sfCCA}}}^{\mathsf{G}_3}(k) &= \Pr[\mathsf{G}_{\mathcal{SE}}^{\mathcal{A}_{\text{PH-sfCCA}}}(k,0,0) = 1] - \Pr[\mathsf{G}_{\mathcal{SE}}^{\mathcal{A}_{\text{PH-sfCCA}}}(k,0,1) = 1]
\end{aligned}$$

We will bound G_1 and G_3 by an adversary in the INT-sfCTXT game and G_2 by an adversary in the PH-CPA game. We begin with G_2. Consider the following adversary $\mathcal{A}_{\text{PH-CPA}}$ defined in the upper half of Figure 5.3.

This adversary simulates the game G_2 with the same value of b and uses the claimed resources. Note that the correctness of the scheme is required here as otherwise the messages returned by $\mathcal{A}_{\text{PH-CPA}}$ might be different from the ones an incorrect scheme might return.

To bound the advantage in G_1 consider the adversary $\mathcal{A}_{\text{INT-sfCTXT}}$ defined in the lower half of Figure 5.3. This adversary simulates the game G_1 up until the first time PH-sfCCA sends an out-of-sync decryption query that would decrypt to something other than $\perp$, again under the assumption that the scheme is correct. This is the same scenario as in the proof of 4.12: if such a query occurs, than the constructed adversary wins the integrity game. So it does not matter what happens after such a query and the stated bound holds. Similarly, we can define an adversary for the game G_3, which is identical to the adversary above except that lines 8, 9 and 10 are deleted. □

Figure 5.4 shows an overview of the most important relations regarding progress-hiding. The important take-away from this chapter is that the notion often used for secure channels, LoR-sfAE, does not provide progress-hiding and therefore is not sufficient for schemes where this is a desired property. Instead, $-sfAE should be the goal of such schemes.

Chapter 6

Analysis of TLS

Having established the new notion \$-sfAE and progress-hiding as a reason why channels might want to achieve it, let us take a look at some real-world protocols so see if they succeed at this goal. Specifically, we analyse the protocols TLS 1.3 and 1.2. These protocols use a stateless scheme as an underlying primitive and construct a channel using this scheme. The requirements on this underlying scheme are a little different from the schemes we have considered so far, so we begin by introducing them.

6.1 Nonce-based schemes with additional data

A problem with randomized schemes is that it is hard to create good randomness. Bad randomness can lead to insecure implementation of encryption schemes, see for example [GM05]. One way to avoid this problem is to introduce nonces, first formalized in [RBBK01]. In nonce-based encryption schemes, the encryption and decryption algorithms take a nonce as an additional input. These schemes should provide security under the assumption that nonces are never reused. So implementations only have to ensure that nonces are never repeated, instead of producing randomness. Channels will often use a sequence number as (part of) the nonce, simultaneously ensuring stateful correctness and providing unique nonces.

Another problem that comes up in real-world schemes is the need for plaintext information sent along with the ciphertext to ensure transmission and parsing of the ciphertext. We do not require privacy for this information but we wish to provide integrity. So we want a scheme which provides integrity for an additional input its for encryption and decryption algorithms. This was first formalized in [Rog02].

Combining these extensions give us nonce-based encryption schemes with additional data.

Definition 6.1
A nonce-based symmetric encryption scheme with additional data $\mathsf{AEAD} := (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}) := (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, l)$ *for* $\mathcal{K}, \mathcal{M}, \mathcal{C}, \mathcal{N}, \mathcal{H} \subset \{0,1\}^*$ *consists of four algorithms. The algorithm* KGen *takes a security parameter k and returns a key* $K \in \mathcal{K}$*. The algorithm* Enc *takes a key K, a nonce* $n \in \mathcal{N}$*, a plaintext* $m \in \mathcal{M}$ *and additional data* $h \in \mathcal{H}$ *and returns a ciphertext* $c \in \mathcal{C}$*. The algorithm* Dec *takes a key, a nonce, a ciphertext and a header and returns a plaintext. The length function l takes a the length of a plaintext and returns an integer. We call such a scheme correct if the decryption of a ciphertext using the same nonce and header that were used for encryption results in the original message.*

We wish to require $-AE security from the underlying scheme. The original definition of $-AE however does not include nonces and additional data. So we now present a slightly adjusted definition. The adversary gets to provide the nonce and the additional data in all encryption and decryption queries. Checks on a ciphertexts are replaced by checks on nonce, ciphertext and additional data. We also add the restriction that each nonce can only be use once in an encryption query. This is commonly referred to as AEAD security.

Definition 6.2
The $-AE *security of a nonce-based encryption scheme with additional data* AEAD *with security parameter k against adversaries with runtime t using q queries totalling μ bits is defined by* $\mathsf{Sec}^{\text{\$-AE}}_{\mathsf{AEAD}}(k,t,q,\mu)$ *using the game* $\text{\$-AE}_{\mathsf{AEAD}}(k,b)$ *depicted in Figure 6.1. The input for encryption queries is a nonce* $n \in \mathcal{N}$*, a plaintext* $m \in \mathcal{M}$ *and additional data* $h \in \mathcal{H}$*. The input to the decryption query is the same except the message is replaced by an arbitrary string c. The input to the finalization query is a bit d.*

6.2 TLS 1.3

TLS 1.3, introduced in 2018 and defined in [Res18], is the latest version of the TLS protocol. We prove that the record layer of TLS 1.3 (with some restrictions) fulfils $-sfAE provided that the underlying scheme fulfils $-AE, a further indication that TLS 1.3 is correctly considered to be very secure.

Next to the symmetric record layer, TLS 1.3 also includes a handshake protocol to establish and refresh symmetric key material. As this thesis is only concerned with symmetric schemes, we assume that the necessary symmetric key material has been shared between the two parties. This means that the handshake to begin the session or to update the key has taken place and our analysis does not cover vulnerabilities that result from attacks against the handshake or the key generation. It is possible to formalize symmetric encryption schemes which can update their keys and define security notions which protect against attacks which go over multiple key phases, see

```
Game $-AE_{AEAD}(k, b)$

proc $\mathsf{Init}_{\$\text{-AE}}()$
  $S \leftarrow \varnothing$
  $N \leftarrow \varnothing$
  $K \leftarrow\$ \mathsf{KGen}(k)$
  return $k$

proc $\mathsf{Enc}_{\$\text{-AE}}(n, m, h)$
  if $n \in N$
    $c_b \leftarrow \perp$
  else
    $c_0 \leftarrow\$ \{0,1\}^{l(|m|)}$
    $c_1 \leftarrow \mathsf{Enc}_K(n, m, h)$
    $N \leftarrow N \cup \{n\}$
    $S \leftarrow S \cup \{(n, c_b, h)\}$
  return $c_b$

proc $\mathsf{Dec}_{\$\text{-AE}}(n, c, h)$
  if $b = 0$ or $(n, c, h) \in S$
    $m \leftarrow \perp$
  else
    $m \leftarrow \mathsf{Dec}_K(n, c, h)$
  return $m$

proc $\mathsf{Fin}_{\$\text{-AE}}(d)$
  return $d$
```

Figure 6.1: The game for $-AE

[GM17]. Their ideas could presumably be combined with $-security to give a better formal approximation of TLS.

We also assume that the keys (in particular the nonce key described below) are sampled randomly from all possible strings of a specific length. We therefore ignore attacks which attack the key generation algorithms.

The final caveat is in regards to the headers of the records. A TLS 1.3 record is defined by

```
struct {
  ContentType opaque_type=application_data;
  ProtocolVersion legacy_record_version=0x0303;
  uint16 length;
  opaque encrypted_record[TLSCiphertext.length];
} TLSCiphertext;
```

As the headers are not encrypted, an adversary can trivially distinguish them from random strings generated by a $-type encryption oracle. However the headers do not really leak any information to the adversary: the headers opaque_type and legacy_record_version are constant and only necessary for legacy reasons to ensure that the record looks like a TLS 1.2 record. The header length gives the length of encrypted_record. This is necessary as in reality, the ciphertext is delivered as a stream of bits. Our formalization of

Scheme $\mathcal{SE}$

```
proc $\mathsf{KGen}_{\mathcal{SE}}(k)$
    $K \leftarrow\$ \mathsf{KGen}_{\mathsf{AEAD}}(k)$
    $\kappa \leftarrow\$ \{0,1\}^N$
    $i \leftarrow 0$
    $j \leftarrow 0$
    $sync \leftarrow 1$
    return $((K,\kappa), i, (j, sync))$

proc $\mathsf{Enc}_{\mathcal{SE}}((K,\kappa), m, i)$
    $i \leftarrow i+1$
    $n \leftarrow \kappa \oplus i$
    $ad \leftarrow h \| l(|m|)$
    $c \leftarrow \mathsf{Enc}_{\mathsf{AEAD}}(K, n, m, ad)$
    return $(c, i)$

proc $\mathsf{Dec}_{\mathcal{SE}}((K,\kappa), m, (j, sync))$
    if $sync = 0$
        $m \leftarrow \perp$
    else
        $j \leftarrow j+1$
        $n \leftarrow \kappa \oplus j$
        $ad \leftarrow h \| |c|$
        $m \leftarrow \mathsf{Dec}_{\mathsf{AEAD}}(K, n, c, ad)$
        if $m = \perp$
            $sync \leftarrow 0$
    return $(m, j, sync)$
```

Figure 6.2: The abstract scheme capturing the record layer of TLS 1.3

security does not make any attempt to hide the length of ciphertexts, so this header does also not leak any new information to the adversary. Note however that the discrepancy between atomic security definitions and stream-based real-world schemes can lead to problems, see for example [FGMP15]. We ignore these issues. All together we therefore assume the headers are a part of the transmission, not to be part of the ciphertext and analyse just the security of an encrypted_record.

So the scheme $\mathcal{SE}$ we are actually analysing is the scheme which transforms a TLSInnerPlaintext into a TLSCiphertext.encrypted_record using the underlying scheme AEAD as described in [Res18]. This scheme, defined in Figure 6.2, works as follows: it maintains counters i and j for the number of messages encrypted and decrypted. A key actually consists of two keys: a key K for AEAD and a random string κ which has the length N of the nonce input to AEAD. The encryption of $\mathcal{SE}$ is defined as the encryption of AEAD using $\kappa \oplus i$ as the nonce and $h \| l(|m|)$ as the additional data for $h = application\_data \| 0x0303$. The decryption is the inverse of this. If a decryption ever fails (so returns $\perp$), then the connection is ended. We model this with a flag *sync*. If a decryption ever outputs $\perp$, then *sync* is set to 0 and all future decryption queries are answered with $\perp$.

We now prove that $\mathcal{SE}$ fulfils \$-sfAE given that the underlying scheme AEAD fulfils \$-AE.

Theorem 6.3
Let $\mathcal{SE}$ be the scheme defined in Figure 6.2. There exists a constant c so that for any k, t, q and μ we have

$$\mathsf{Sec}^{\text{\$-sfAE}}_{\mathcal{SE}}(k,t,q,\mu) \leq \mathsf{Sec}^{\text{\$-AE}}_{\mathsf{AEAD}}(k,t+cq,q,\mu).$$

Proof Let $\mathcal{A}$ be an adversary against $\mathcal{SE}$ in the \$-sfAE game and N be the length of the nonce input for AEAD. We construct an adversary $\mathcal{A}_{\text{\$-AE}}$ against AEAD in the \$-AE game follows the construction of $\mathcal{SE}$ but replaces encryptions and decryptions by calls to its \$-AE game.

```
$\mathcal{A}_{\text{\$-AE}}$

proc $\mathsf{Init}_{\text{\$-sfAE}}()$
  $\kappa \leftarrow \{0,1\}^N$
  $i \leftarrow 0$
  $j \leftarrow 0$
  $sync \leftarrow 1$
  $d' \leftarrow 0$
  return $\mathsf{Init}_{\text{\$-AE}}()$

proc $\mathsf{Enc}_{\text{\$-sfAE}}(m)$
  $i \leftarrow i+1$
  $n \leftarrow \kappa \oplus i$
  $ad \leftarrow h \| l(|m|)$
  $c_i \leftarrow \mathsf{Enc}_{\text{\$-AE}}(n,m,ad)$
  return $c_i$

proc $\mathsf{Dec}_{\text{\$-sfAE}}(c)$
  if $sync = 0$
    $m \leftarrow \perp$
  else
    $j \leftarrow j+1$
    $n \leftarrow \kappa \oplus j$
    $ad \leftarrow h \| |c|$
    $m \leftarrow \mathsf{Dec}_{\text{\$-AE}}(n,m,ad)$
    if $j > i$ or $c \neq c_j$
      $sync \leftarrow 0$
      if $m \neq \perp$
        $d' \leftarrow 1$
  return $m$

proc $\mathsf{Fin}_{\text{\$-sfAE}}(d)$
  return $\mathsf{Fin}_{\text{\$-AE}}(d$ or $d')$
```

This adversary uses the stated resources. We claim that it simulates the \$-sfAE game for $\mathcal{A}$ with the same value of b up until the second out-of-sync query and if the simulation fails, then it wins the \$-AE game. The return value of encryption queries sent by $\mathcal{A}$ are distributed the same in an \$-sfAE game and in the game simulated by $\mathcal{A}_{\text{\$-AE}}$ for the same value of b, as the encryption queries are the same for stateful and stateless AE games.

In-sync decryption queries from $\mathcal{A}$ become decryption queries which are contained in S for the \$-AE game, so they are suppressed correctly. Now consider the first out-of-sync query made by $\mathcal{A}$. We claim that the combination of nonce and ciphertext is new in the \$-AE game, so the query is not suppressed. Indeed, if the ciphertext is entirely new, then this certainly holds. If the ciphertext is not new, then the value of j used by $\mathcal{A}_{\text{\$-AE}}$ to create the nonce in the decryption query will not match the value of i used for the encryption of the ciphertext.

If $b = 0$ or if the underlying scheme fulfils integrity for this query, then the return value of this query will be $\perp$. This means that every query after this will be suppressed in both an \$-sfAE game and by $\mathcal{A}_{\text{\$-AE}}$. So in this case the game is simulated in its entirety with the same value of b, and (since d' is never set), the output is also the same. So the stated bound holds.

Now assume that the return-value to the first out-of-sync decryption query is not $\perp$. This can only occur if $b = 1$ and the underlying scheme fails to have integrity. In this case, the simulation may be wrong, as every query is suppressed by $\mathcal{A}_{\text{\$-AE}}$ even though $\mathcal{A}$ might expect queries to be answered in the \$-sfAE game. But in this case d' is set to 1 and $\mathcal{A}_{\text{\$-AE}}$ will always correctly output 1. So in this case the stated bound also holds. $\square$

Applying Theorem 5.10 we see that the record layer of TLS 1.3 succeeds at providing progress-hiding.

6.3 TLS 1.2

We now turn to TLS 1.2, which was introduced in 2008 and is defined in [DR08]. Specifically, we show that it in general does not fulfil PH-sfCCA, and therefore also does not fulfil \$-sfAE, even when using an underlying \$-AE scheme. Note that TLS 1.2 has known vulnerabilities which are far more severe than the lack of progress-hiding, but it is interesting to see that the improvements made in TLS 1.3 have the positive side-effect of ensuring progress-hiding.

We can mostly make similar assumptions to above, namely that we only consider the record layer and assume that keys are generated sufficiently randomly. We cannot however make the same assumptions for the headers. The headers of TLS 1.2 are mostly the same (with a different version number) with one important difference: the field opaque_type is used. It can contain the following values:

```
enum {
  change_cipher_spec(20), alert(21), handshake(22),
  application_data(23), (255)
} ContentType;
```

The values change_cipher_spec and handshake are not relevant for the record layer. The values alert and application_data can however both occur. So an adversary can possibly learn something about the communication from the header, namely it can tell alert messages from real messages. While this leak does not seem like a huge issue, it is certainly not as justifiable to ignore the headers in the analysis. Including these headers would of course make it impossible to achieve \$-sfAE. It turns out however that there are even bigger issues relating to the use of nonces.

TLS 1.2 supports three types of underlying schemes: stream ciphers, block ciphers and AEAD ciphers. Block and stream ciphers require the combination of an encryption scheme and a MAC. The analysis of this combination is outside the scope of this thesis. We will only make one small remark: the results on combining encryption schemes and MAC algorithms mostly use LoR oracles. They do not necessarily hold if we use \$-type oracles. For example the Encrypt-then-Mac paradigm will not preserve \$-CPA security from the encryption scheme unless the MAC outputs are distributed randomly, something not automatically required from a MAC scheme.

Instead, we will concentrate on AEAD ciphers, which are considered more secure for other reasons as well. The input to these is similar to the underlying schemes used for TLS 1.3 with one important exception: the generation of nonces is not specified. Instead each cipher has to specify this itself. This can involve a fixed value, which is generated during the handshake and an explicit value which is prepended to the ciphertext. The prepended explicit value can therefore leak information. Depending on the cipher, this can prevent the scheme from achieving PH-sfCCA.

There are three recommended AEAD ciphers for TLS 1.2: AES-GCM, AES-CCM and ChaCha20. AES-GCM and AES-CCM handle their nonce in the same way: Each nonce is 12 bytes long. The first 4 bytes are fixed and generated during the handshake. The only requirement on the remaining 8 bytes, which are sent along with the ciphertext, is that they are unique over a session. The specification explicitly states the sequence number as an option for this. Doing this of course implies that the channel cannot achieve PH-CPA: an adversary for progress-hiding can simply check the value of this counter by sending an encryption query after sending a progress query. Presumably one could however prove that TLS 1.2 fulfils LoR-sfAE assuming the same restrictions and using the same ideas as for above for TLS 1.3.

Note that sending the sequence number along with the ciphertext is not only insecure, but also unnecessary, as the receiver is maintaining a matching counter anyway. So it fairly easy (as it is done in TLS 1.3) to just leave it out. Indeed, the other ciphersuite recommended for TLS 1.2 does this. In ChaCha20, the length of the explicit part of the nonce is zero, so nothing is sent along with the ciphertext. Instead, just as in TLS 1.3, the sequence number is XORed with a key agreed upon during the handshake. Following the analysis of TLS 1.3, one could show that (up to additional problems with ignoring the headers) the \$-sfAE security of TLS 1.2 is bounded by the \$-AE security of ChaCha20. Of course, as we saw above, ignoring the headers is not as valid as it is for TLS 1.3.

Chapter 7

General notions of correctness

7.1 Predicates

So far we have assumed that our encryption scheme is running over a reliable network protocol like TCP. This allowed us to assume that the messages arrive in the order they were sent. There are however schemes which run over network protocols where such assumption do not hold, like UDP. This means that messages can be reordered, duplicated or dropped by the network instead of by an adversary. So we cannot just abort the channel when this occurs. Instead we need to decide when this should be viewed as harmless networks effects, and therefore decrypt correctly, and when it should be attributed to tampering and therefore be rejected. There is no single correct answer for this question, instead the answer depends on the network, the scheme and the use-case. For example, in [KPB03] and [BHMS15] a hierarchy of authentication notions are defined, which provide increasing levels of protection.

Instead of defining specific levels and analysing these individually, we now introduce predicates, which can encode many different notions of correctness and integrity. We then define generic security notions based on predicates and prove relations for these notions. Specifying a predicate then immediately gives us the appropriate security notions and relations.

This idea of such predicates is not new. In [RZ18] ”indistinguishability up to correctness” is defined. Correctness is defined there using a predicate, which is an efficiently computable function which takes as input the transcript of inputs and outputs to the privacy game and decides whether the result of query is fixed, e.g. required by correctness. While very general, we run in to some problems if we try to use the same definition for our predicates.

We namely wish to define not only a privacy game but also a correctness game. The notions of stateless and stateful correctness used so far are generally fairly easy to prove, so it is not necessary to define a correctness game. If we consider more complex correctness notions, it can however be useful to define a security notion which ensures that a scheme is correct. This also allows us to quantize schemes which do not fulfil correctness perfectly, but for which it is very hard for an efficient adversary cause a false decryption to occur. Next to the correctness game, we also wish to define a progress-hiding game and a combined correctness/privacy game. We see in a moment why we do not need a separate integrity game. The same predicate should be used to define each of these games. This means however that we cannot use predicates which see the transcript of the game (so the queries sent by the adversary and the returned values), as the possible queries and their answers are different for each of these games. Instead we define predicates which use the transcript of the encryption scheme, so the inputs and outputs of the encryption and decryption algorithms. This actually matches the intuition behind most concrete correctness notions. For many notions, like $-CPA, this is equivalent. There can however be subtle differences, which lead us to the next issue with defining predicates.

Imagine a notion of correctness where the encryption of a specific message m causes all remaining queries to not be bound by correctness and a corresponding privacy game using a LoR oracle. If the adversary sends m as one of the messages in a LoR query, then later messages are either bound by correctness or not depending on which messages was encrypted. As the privacy game should suppress queries which are bound by correctness, this could allow the adversary to win the privacy game by checking whether a later query is suppressed or not. So a scheme which fulfils this correctness notion might inherently not be able to achieve privacy. Note that this issue only arises because the predicate is defined on the transcript of the scheme. If the predicate used the transcript of the game instead, it would not be possible to define this correctness notion.

So how do we deal with this issue? One could attempt to define a more general privacy game, which can account for these issues. This is however fairly complicated and it turns out that this issue does not occur for real-world correctness notions. So we instead use an idea from [FGJ20] and restrict the predicate to only see the ciphertext portions of the transcript, so the ciphertexts output by the encryption algorithm and the ciphertexts input to the decryption algorithm. This ensures that for the privacy game, the adversary has access to the entire transcript and can therefore calculate the predicate itself. This ensures that for schemes which provide privacy, the predicate cannot depend on the value of b in the privacy game. Following another idea from [FGJ20] we also include the output of the predicate in the transcript for decryption queries. This makes it easy (and efficient) to define some concrete correctness notions.

The idea of the correctness game is simple. The adversary has access to an encryption and decryption oracle and wins the game if it can input a ciphertext which does not decrypt the way that the predicate says it should. There is however still a problem. The predicate only determines whether the decryption is bound by correctness and does not provide the result of the correct decryption. To fix this we make an additional assumption on our predicate, namely that only ciphertexts which were output by the encryption algorithm can be correct. Such ciphertexts have a natural correct decryption, namely the plaintext which was input to the encryption algorithm. In other words, no predicate can be "more correct" than stateless correctness.

There is another important restriction we need to make, namely ciphertext uniqueness. This means that no two ciphertexts output by the encryption algorithm can ever be equal. Syntactical, this is required to ensure that operations of look-up tables and arrays are well-defined. But intuitively this restriction also makes sense: If two different messages encrypt to the same ciphertext, it is not clear which of the two messages is the "correct" decryption. Even if the same plaintext is encrypted twice, it needs to result in different ciphertexts. Otherwise the behaviour of for example replay-protection is unclear. We refer again to [FGJ20] for further discussion of how to handle these issues. From here on we implicitly assume that all of the schemes fulfil ciphertext uniqueness.

Both of these restriction could be relaxed in exchange for some extra formalism. One could extend the predicates to include a second function, which takes as input a ciphertext which the original predicate deemed to be correct and outputs the value of the correct decryption. This second function would also receive as input the entire transcript of the encryption scheme. All the results of this and the next chapters could presumably be extended for such predicates, but the added syntactical overhead would make it fairly tedious.

Having sufficiently restricted our predicates, we now present an extension that allows them to ensure not only correctness but also integrity. This is possible by letting the predicate output three possible values: The output 1 means the decryption should be correct, the output 0 means it should be $\perp$ and the output $\lightning$ means that the predicate does not specify the output. This will allow us to define a single game which can ensure correctness, integrity or a combination of both and will give some interesting insights into the relation between the two.

Definition 7.1
A transcript T is an array with entries of the form (enc, c) and (dec, c, p), where c is a string and $p \in \{1, 0, \lightning\}$. Let $\mathcal{T}$ be the set of all transcripts. A predicate is an efficiently computable function

$$\varphi : \mathcal{T} \times \mathcal{C} \to \{1, 0, \lightning\}$$

with the property that if $\varphi(T, c) = 1$ then $(enc, c) \in T$.

A predicate which only outputs 1 and $\lightning$ is called a correctness predicate, one which only outputs 0 and $\lightning$ an integrity predicate. A predicate which only outputs 0 and 1 is called fully-specified. For a correctness predicate, we can define the corresponding ciphertext integrity predicate by replacing every output of 1 with $\lightning$ and every $\lightning$ with 0. For any predicate φ define its correctness portion φ^{cor} to be the predicate where we replace every 0 with $\lightning$. Similarly, for the integrity portion φ^{int} replace every 1 with $\lightning$.

The transcript T represents a series of ciphertexts which were either an output of the encryption algorithm or an input to the decryption algorithm, along with the previous decisions of a predicate φ. For a ciphertext c, φ decides what the output of the decryption of c should be given the current transcript.

In order to prove relations between security notions defined using predicates, we need to understand how predicates can relate to each other.

Definition 7.2
Let φ and ψ be predicates. We say that φ and ψ contradict each other if there exists an input so that they specify different values, so

$$\exists T, c : \lightning \neq \varphi(T, c) \neq \psi(T, c) \neq \lightning$$

If they do not contradict each other, then we define the union $\varphi \cup \psi$ by

$$\varphi \cup \psi(T, c) = \begin{cases} \varphi(T, c) & \textit{if } \varphi(T, c) \neq \lightning \\ \psi(T, c) & \textit{else} \end{cases}.$$

Finally, if ψ is equal to φ for all values for which it is specified, so if

$$\forall T : \psi(T, c) \neq \lightning \Rightarrow \psi(T, c) = \varphi(T, c),$$

then we say that φ implies ψ and write $\psi \subset \varphi$.

A correctness notion can always be combined with its corresponding ciphertext integrity predicate, which results in a fully-specified predicate.

We make a final remark on the capabilities of our predicates. For most practical applications, it would suffice to only consider correctness predicates and define an integrity game which ensures the corresponding ciphertext integrity. Next to the interesting generalizations and perspectives on the interplay of correctness, integrity and privacy, there are some practical advantages to our approach. One advantage is the possibility of combining integrity and correctness into a single game. We will see in Chapter 9 how this helps with the analysis of DTLS.

Another interesting feature is being able to define predicates which have some correctness, some integrity but also some unspecified values. In practice, it is not clear if this is desirable, as we generally wish every output of a scheme to be correct or suppressed by integrity. One problem which one could try to overcome with this flexibility, is the desire to extend the error set beyond just $\perp$. Instead of allowing unspecified behaviour it is however certainly better to specifically formalise error sets, which has been done for example in [BDPS14].

7.2 Concrete correctness notions

We now show how our predicates can be used to define common correctness and integrity notions. In particular, we define predicates corresponding to the hierarchy from [KPB03] and [BHMS15]. This hierarchy starts with stateless integrity and adds replay, reorder and dropping protection one by one to finally end up with stateful integrity. We also define predicates which rely on windows, similar to those used in [FGJ20] to analyse QUIC and DTLS. An overview of the predicates defined here can be found in Figure 7.1.

We define these predicates as fully-specified predicates which provide correctness and integrity. Instead of an increasing heirarchy, we define individual predicates for each security goal. Using the results below, we can then combine these primitives to create predicates for multiple goals. This will allow us to define the commonly used correctness notions, but also allows us to look at different goals separately or create exciting new combinations!

Before we start defining predicates we introduce some helpful syntax. Let T_{enc} and T_{dec} be the arrays containing only the encryption or decryption entries of the transcript and let $ind_{enc}(c)$ be the index of c in T_{enc}. We will be careful to only use this if we know that that (enc, c) is indeed contained in T to avoid defining what the output should be for elements not contained in T. Finally, let

$$latest(T) := \max\{ind_{enc}(c) \mid (enc, c) \in T \text{ and } (dec, c, 1) \in T\},$$

be the index of the latest ciphertext which was accepted. Latest here refers to the order the ciphertexts were encrypted, so this is not necessarily the last ciphertext decrypted. Note all of these operations can be performed efficiently given a transcript.

Stateless and stateful integrity

The stateless predicate φ_{sl} rejects ciphertexts which were not encrypted and is defined as

$$\varphi_{sl}(T,c) = \begin{cases} 1 & \textbf{if } (enc,c) \in T \\ 0 & \textbf{else} \end{cases}$$

The stateful predicate rejects ciphertexts if the sequence of ciphertexts decrypted is not a subsequence of encrypted ones. For this let $j = |T_{dec}|$ be the number of decrypted ciphertexts and define

$$\varphi_{sf}(T,c) = \begin{cases} 1 & \textbf{if } j \leq |T_{enc}| \text{ and } \forall i < j : T_{dec}[i] = T_{enc}[i] \text{ and } c = T_{enc}[j] \\ 0 & \textbf{else} \end{cases}$$

Acceptance windows

An acceptance-window rejects ciphertexts if they are outside a window w_p before and w_f after the latest ciphertext accepted. The values w_p and w_f can be constant, but they can also be function which decide dynamically for each ciphertext how large the window should be. The predicate is defined as

$$\varphi_{(w_p,w_f)}(T,c) = \begin{cases} 1 & \textbf{if } (enc,c) \in T \\ & \text{ and } -w_p(c) \leq ind_{enc}(c) - latest(T) \leq w_f(c) \\ 0 & \textbf{else} \end{cases}$$

If we set $w_p = w_f = \infty$ then we have the stateless ciphertext integrity predicate.

Replay protections

The replay protection predicate φ_{rep} rejects ciphertext that have been accepted before. It is defined as

$$\varphi_{rep}(T,c) = \begin{cases} 1 & \textbf{if } (enc,c) \in T \text{ and } (dec,c,1) \notin T \\ 0 & \textbf{else} \end{cases}$$

Reorder protection

The reorder protection predicate φ_{ord} rejects a ciphertext if its index is not strictly greater than $latest(T)$. This means that the indexes of the decrypted ciphertexts will be strictly increasing. It is defined as

$$\varphi_{ord}(T,c) = \begin{cases} 1 & \textbf{if } (enc,c) \in T \text{ and } ind_{enc}(c) > latest(T) \\ 0 & \textbf{else} \end{cases} .$$

Dropping protection

The dropping protection predicate φ_{drop} rejects a ciphertext c if every ciphertext which was encrypted before c has not been decrypted yet and is defined as

$$\varphi_{drop}(T,c) = \begin{cases} 1 & \textbf{if } (enc,c) \in T \\ & \text{ and } \forall (enc,c') \in T : ind_{enc}(c') \leq ind_{enc}(c) \Rightarrow (dec,c',1) \in T \\ 0 & \textbf{else} \end{cases}$$

Note that all of the predicates defined above (except for φ_{sf}) are forgiving, so a rejected query does not cause all future queries to be rejected. One could also define strict versions of each predicate, where a single rejected query causes all further queries to be rejected. To do this formally, one would simply add a check $\nexists c : (dec,c,0) \in T$ to a predicate.

One can also define predicates somewhere in between forgiving and strict. For example, one could allow a certain number of errors before rejecting every query. Another example is the notion of robustness defined in [FGJ20], which ensures that the output of correct queries is not changed by the fact that there are forgery attempts. The behaviour for previously encrypted ciphertexts is then defined by their notion of correctness, which allows strict notions. So their schemes can be forgiving for forgeries but strict for other attacks. Their notions of correctness, robustness and integrity can all be defined using predicates.

Theorems 7.6 and 7.11 below show that fulfilling the union of two predicates is equivalent to fulfilling both of the predicates individually. This allows us to take the union of predicates to achieve multiple types of protection. We cannot however take the union of fully-specified predicates directly, as they are either equal or contradict each other. Instead, the way to combine predicates is to take the union of their integrity portions. One can then turn the combined integrity predicate back into a fully-specified one by replacing the remaining $\lightning$'s with 1's.

Note that it is important to do this with the integrity portion, not the correctness portion. While we can take the union of correctness portions, the resulting notions are not useful. For example, combining the replay correctness and dropping correctness predicate results in a predicate which causes queries which are a replay or a drop to be correct and only suppresses queries which are simultaneously a replay and a dropping attack.

We now list some relations between the predicates defined above. We have $\varphi^{int}_{rep} \subset \varphi^{int}_{ord}$, so (by Theorem 7.6) a scheme which protects against reorder attacks automatically protects against replay attacks. Also for any predicate φ we have $\varphi^{int}_{sl} \subset \varphi^{int}$, or in words every predicate provides protection

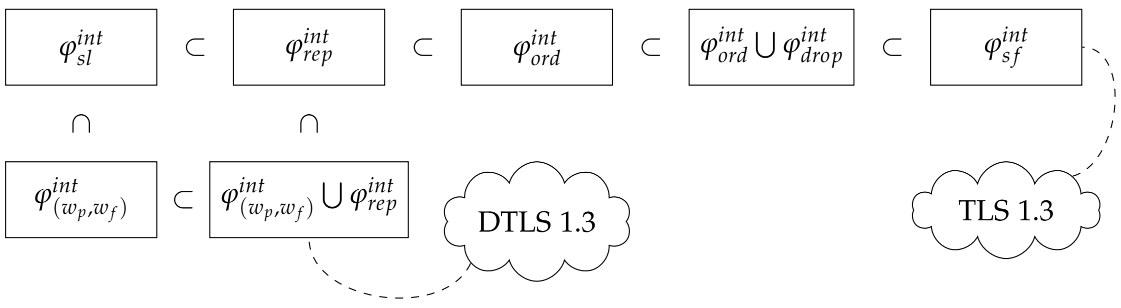

Figure 7.1: Relations between common predicates. The upper row represents the classical hierarchy from stateless to stateful notions defined in [KPB03]. Replacing the integrity predicates with the corresponding correctness predicates would reverse the implications.

against forgeries. This is a direct consequence of the assumption that only encrypted ciphertexts can be correct.

Another interesting example of this is the combination of φ_{ord} and φ_{drop}. A ciphertext is only accepted if every ciphertext encrypted before it has been decrypted. This is equivalent to $\varphi_{(0,1)}$ and is the forgiving version of φ_{sf}. One can view this as a formal confirmation that stateful security indeed protects against replay, reorder and dropping attacks.

In Chapter 9 we will use the combination of replay-protection and an acceptance window in order to analyse DTLS.

7.3 Security notions using predicates

We define three notions using predicates. We start with the fulfilment notion, which ensures that a scheme fulfils a predicate. Next we define a privacy notion which ensures privacy "up to a predicate", which means that decryption queries which are specified by the predicate are suppressed. Finally, we define a combined notion, which ensures both fulfilment and privacy.

7.3.1 The fulfilment notion

An adversary wins this game if it can send a decryption query which is specified by the predicate and decrypts to the wrong value. To check whether a decryption is correct, we save each message encrypted along with the resulting ciphertext. By the assumption that only such ciphertexts can be correct, we can use this to check the decryption.

Definition 7.3 (φ-FUL)
For a predicate φ the predicate fulfilment of an encryption scheme $\mathcal{SE}$ with security parameter k against adversaries with runtime t using q queries totalling μ bits

```
Game $\varphi$-$\mathsf{FUL}_{\mathcal{SE}}(k)$

proc $\mathsf{Init}_{\varphi\text{-}\mathsf{FUL}}()$
    $K \leftarrow\!\!\$\ \mathsf{KGen}(k)$
    $T \leftarrow \varnothing$
    $E \leftarrow \varnothing$
    $win \leftarrow 0$
    return $k$

proc $\mathsf{Enc}_{\varphi\text{-}\mathsf{FUL}}(m)$
    $c \leftarrow \mathsf{Enc}_K(m)$
    $T \leftarrow T \| (enc, c)$
    $E \leftarrow E \| \{c : m\}$
    return $c$

proc $\mathsf{Dec}_{\varphi\text{-}\mathsf{FUL}}(c)$
    $m \leftarrow \mathsf{Dec}_K(c)$
    $p \leftarrow \varphi(T, c)$
    $T \leftarrow T \| (dec, c, p)$
    if $p = 0$ and $m \neq \perp$
        $win \leftarrow 1$
    if $p = 1$ and $m \neq E[c]$
        $win \leftarrow 1$
    return $m$

proc $\mathsf{Fin}_{\varphi\text{-}\mathsf{FUL}}()$
    return $win$
```

Figure 7.2: The game for φ-FUL

is defined by $\mathsf{Sec}^{\varphi\text{-}\mathsf{FUL}}_{\mathcal{SE}}(k, t, q, \mu)$ *using the game* φ-$\mathsf{FUL}_{\mathcal{SE}}(k)$ *depicted in Figure 7.2. The input for encryption queries are messages* $m \in \mathcal{M}$ *and the input for the decryption queries are strings* c*.*

For a correctness predicate this game ensures that the scheme is correct. In particular, a scheme fulfils the original stateless or stateful correctness notion if and only if it has perfect security in the fulfilment game for φ^{cor}_{sl} or φ^{cor}_{sf}. For the corresponding ciphertext integrity predicate, this game is equivalent to the appropriate ciphertext integrity game. For example, φ^{int}_{sl}-FUL is equivalent to INT-CTXT. With a combined predicate this game provides correctness and integrity. Note that formally, this last statement requires Theorem 7.11, which shows that the game for a combined predicate is equivalent to the combination of the games for each individual predicate.

An interesting observation is that we cannot define a predicate for plaintext integrity, as this depends on the output of the decryption, which is not known to the predicate. We could however define specific forms of plaintext integrity, for example not forcing integrity for ciphertexts where exactly one bit has been flipped from an original ciphertext. It is not clear whether there is much use for such definitions.

7.3.2 The privacy notion

These games are based on the privacy notions defined in Chapter 2 and 3. The flavors remain the same as before and any decryption query which is specified by predicate is suppressed.

```
Game FL-$\varphi$CCA$_{\mathcal{SE}}(k, b)$

proc Init$_{\text{FL-}\varphi\text{CCA}}()$
    $T \leftarrow \varnothing$
    return Init$_{\text{FL-CPA}}()$

proc Enc$_{\text{FL-}\varphi\text{CCA}}(\overline{m})$
    $c \leftarrow$ Enc$_{\text{FL-CPA}}(\overline{m})$
    $T \leftarrow T \| (enc, c)$
    return $c$

proc Chal$_{\text{FL-}\varphi\text{CCA}}(\overline{m})$
    $c \leftarrow$ Chal$_{\text{FL-CPA}}(\overline{m})$
    $T \leftarrow T \| (enc, c)$
    return $c$

proc Dec$_{\text{FL-}\varphi\text{CCA}}(c)$
    $m \leftarrow$ Dec$_K(c)$
    $p \leftarrow \varphi(T, c)$
    $T \leftarrow T \| (dec, c, p)$
    if $p \neq \lightning$
        $m \leftarrow \bot$
    return $m$

proc Fin$_{\text{FL-}\varphi\text{CCA}}(\overline{d})$
    return Fin$_{\text{FL-CPA}}(\overline{d})$
```

Figure 7.3: The game for FL-φCCA

Definition 7.4 (FL-φCCA)
For a predicate φ the privacy of an encryption scheme $\mathcal{SE}$ up to φ with security parameter k against adversaries with runtime t using q queries totalling μ bits is defined by $\mathsf{Sec}_{\mathcal{SE}}^{\text{FL-}\varphi\text{CCA}}(k, t, q, \mu)$ using the game FL-φCCA$_{\mathcal{SE}}(k, b)$ depicted in Figure 7.3. The input for the decryption queries are strings c. For the remaining queries the inputs are the same as in the FL-CPA *game.*

For a correctness predicate this game is equivalent to the expected privacy notion, so using φ_{sl}^{cor} gives us FL-CCA and using φ_{sf}^{cor} gives us FL-sfCCA. Using an integrity predicate does not really result in a meaningful game. Indeed, if the scheme is correct (for some correctness notion) and we do not suppress correct decryptions, then an adversary can usually achieve a trivial win, following the strategy from Theorem 2.15. Since every useful scheme fulfils some sort of correctness, this game is not very interesting. Note however that for fully-specified predicate every decryption query is suppressed. So in this case the CCA privacy game reduces to the CPA privacy game. This fact is important for the generic composition result below.

7.3.3 The combined notion

This notion provides both privacy up to a predicate φ and the fulfilment of φ. This makes it a little different from the AE games from Chapter 4. Those games provided privacy up to correctness predicate and ensured the fulfilment of the corresponding integrity predicate. Our games use a single predicate, which can be any combination of correctness and integrity,

Game $\mathsf{FL}\text{-}\varphi\mathsf{AE}_{\mathcal{SE}}(k,b)$

```
proc $\mathsf{Init}_{\mathsf{FL}\text{-}\varphi\mathsf{AE}}()$
    $T \leftarrow \varnothing$
    $E \leftarrow \varnothing$
    return $\mathsf{Init}_{\mathsf{FL}\text{-}\mathsf{CPA}}()$

proc $\mathsf{Enc}_{\mathsf{FL}\text{-}\varphi\mathsf{AE}}(\overline{m})$
    $c \leftarrow \mathsf{Enc}_{\mathsf{FL}\text{-}\mathsf{CPA}}(\overline{m})$
    $T \leftarrow T \| (enc, c)$
    $E \leftarrow E \| \{c : m\}$
    return $c$

proc $\mathsf{Chal}_{\mathsf{FL}\text{-}\varphi\mathsf{AE}}(\overline{m})$
    $c \leftarrow \mathsf{Chal}_{\mathsf{FL}\text{-}\mathsf{CPA}}(\overline{m})$
    $T \leftarrow T \| (enc, c)$
    $E \leftarrow E \| \{c : m\}$
    return $c$

proc $\mathsf{Dec}_{\mathsf{FL}\text{-}\varphi\mathsf{AE}}(c)$
    $m \leftarrow \mathsf{Dec}_K(c)$
    $p \leftarrow \varphi(T, c)$
    $T \leftarrow T \| (dec, c, p)$
    if $(p \neq \lightning$ and $b = 0)$
        or $(p = 1$ and $m = E[c])$
            or $(p = 0$ and $m = \perp)$
                $m \leftarrow \varepsilon$
    return $m$

proc $\mathsf{Fin}_{\mathsf{FL}\text{-}\varphi\mathsf{AE}}(\overline{d})$
    return $\mathsf{Fin}_{\mathsf{FL}\text{-}\mathsf{CPA}}(\overline{d})$
```

Figure 7.4: The game for FL-φAE

but provides privacy up to and fulfilment of the same predicate, not two different predicates. This means that this notion cannot capture the combination of privacy and integrity without also ensuring correctness. One could also define a combined game which takes as input two predicates, one for privacy and and one for fulfilment, allowing us to define arbitrary combinations. We see below however that this definition allows us to achieve the combination of correctness, integrity and privacy up to correctness, so we just use this simpler game.

Theorem 7.9 states that this combined game is indeed equivalent to the combination of φ-FUL and FL-φCCA. So for a correctness predicate this game is ensures correctness and privacy. For a fully-specified predicate this game ensures correctness, integrity and CPA-privacy. Combining this with Theorem 7.10, generic composition for predicates, shows that this combination actually also ensures CCA-privacy. By Theorem 7.8 taking FL = \$ gives the strongest version of this notion.

As before, much of the previous analysis of channels uses LoR oracles. So the notion \$-φAE is new for most predicates. One exception can be found in [DFP$^+$20], where the authors define a very specific notion for the analysis of QUIC which uses \$-type oracles. We will see the advantage of using our very general notion in Chapter 9 when analyse the scheme DTLS.

The idea of the combined game is as follows: encryption is the same as in the privacy game. Specified decryption queries are suppressed if $b = 0$ or if the output of the decryption is incorrect, allowing an adversary to win if it can cause the scheme to decrypt incorrectly. Unspecified decryption queries are always returned. In this game we have to make a distinction between decryption queries suppressed by the scheme, so decryptions which output $\perp$, and queries suppressed by the game, which we will represent by ε. If we did not make this distinction, an adversary would not be able to distinguish whether a decryption query bound by correctness was decrypted correctly (resulting in ε) or if the scheme incorrectly returned $\perp$.

Definition 7.5 (FL-φAE)
For a predicate φ the φ-authenticated encryption security of an encryption scheme $\mathcal{SE}$ with security parameter k against adversaries with runtime t using q queries totalling μ bits is defined by $\mathsf{Sec}_{\mathcal{SE}}^{\mathsf{FL}\text{-}\varphi\mathsf{AE}}(k,t,q,\mu)$ using the game $\mathsf{FL}\text{-}\varphi\mathsf{AE}_{\mathcal{SE}}(k,b)$ depicted in Figure 7.4. The input for the decryption queries are strings c. For the remaining queries the inputs are the same as in the FL-CPA *game.*

7.4 Relations between predicate notions

We now prove relations between the notions defined using predicates. These results imply practically all of the theorems proved so far in this thesis and allow us to quickly prove similar results for additional correctness notions. Instead of constructing adversaries and analysing simulations for each new predicate, it suffices to define and compare the relevant predicates. In particular, if we show that one predicate implies another, then almost all of the results we showed in Chapters 3 and 4 about the relation between stateless and stateful correctness hold for these two predicates as well.

We begin by showing that implication of predicates ensures the implication of the corresponding fulfilment notions.

Theorem 7.6
Let φ and ψ be correctness predicates. If $\psi \subset \varphi$ then for any k, t, q and μ we have

$$\mathsf{Sec}_{\mathcal{SE}}^{\psi\text{-}\mathsf{FUL}}(k,t,q,\mu) \leq \mathsf{Sec}_{\mathcal{SE}}^{\varphi\text{-}\mathsf{FUL}}(k,t,q,\mu)$$

As the φ_{sl}^{cor} implies every other correctness predicate, this theorem shows that stateless correctness implies every other correctness notion. This implication is reversed for the integrity notions, so every integrity notion implies φ_{sl}^{int}.

Proof Let $\mathcal{A}$ be an adversary for the ψ-FUL game and consider the adversary for the φ-FUL game which runs $\mathcal{A}$ and forwards its queries. This simulates the ψ-FUL game. Now consider the a query where *win* is set to 1 in the simulated game. This is decryption query for ciphertext c so that $\psi(T,c) \neq \lightning$ and the actual decryption $\mathsf{Dec}_{\mathcal{SE}}(c)$ is not the value specified by ψ. But as $\psi \subset \varphi$ and $\psi(T,c) \neq \lightning$, we know that $\psi(T,c) = \varphi(T,c)$. So the decryption also does not match the value given by φ and *win* is also set to 1 in the actual φ-FUL game. □

Note that the converse of this theorem is not quite true. Ideally we would like to prove that if $\psi \not\subset \psi$, so if there exists an input so that $\psi(T,c) \neq \lightning$ and $\varphi(T,c) \neq \psi(T,c)$ either specifies differently or not at all, then φ-FUL $\not\Leftarrow$ ψ-FUL. If an adversary can cause such an input to occur, then sending it will indeed give an attack against the φ-FUL game. Just because such a transcript exists it is however not given that an efficient adversary can cause this transcript to occur. So instead of demanding $\psi \subset \varphi$ above, it would suffice to demand that no adversary can cause a transcript to occur which contradicts $\psi \subset \varphi$, even if such a transcript may exist. This would requires some effort to formalize and for all of the concrete correctness notions defined above, contradictions can always be found efficiently. For example, an out-of-sync ciphertext will create a contradiction between φ_{sl} and φ_{sf}.

We now show that for privacy notions, the implication between predicates is reversed. This is not surprising, as a predicate which specifies more makes it "easier" to win the fulfilment game, but "harder" to win the privacy game.

Theorem 7.7
Let φ and ψ be correctness predicates so that $\psi \subset \varphi$. There exists a constant c so that for any k, t, q and μ we have

$$\mathsf{Sec}^{\mathsf{FL}\text{-}\varphi\mathsf{CCA}}_{\mathcal{SE}}(k,t,q,\mu) \leq \mathsf{Sec}^{\mathsf{FL}\text{-}\psi\mathsf{CCA}}_{\mathcal{SE}}(k,t+cq,q,\mu)$$

Applying this theorem to the φ^{cor}_{sl} and φ^{cor}_{sf} shows Theorem 3.2, which stated that FL-sfCCA $\Rightarrow$ FL-CCA. Recall that if we choose φ to be a fully-specified predicate, then every decryption query is suppressed, so this theorem also shows that FL-φCCA $\Rightarrow$ FL-CPA for any predicate.

Proof Let $\mathcal{A}_\varphi$ be an adversary for the FL-φCCA game and let $\mathcal{A}_\psi$ be the following adversary for the FL-ψCCA game:

$\underline{\mathcal{A}_\psi}$

```
proc $\mathsf{Init}_{\text{FL-}\varphi\text{CCA}}()$
    $T \leftarrow \varnothing$
    return $\mathsf{Init}_{\text{FL-}\psi\text{CCA}}()$

proc $\mathsf{Enc}_{\text{FL-}\varphi\text{CCA}}(m)$
    $c \leftarrow \mathsf{Enc}_{\text{FL-}\psi\text{CCA}}(m)$
    $T \leftarrow T \| (enc, c)$
    return $c$

proc $\mathsf{Enc}_{\text{FL-}\varphi\text{CCA}}(m)$
    $c \leftarrow \mathsf{Enc}_{\text{FL-}\psi\text{CCA}}(m)$
    $T \leftarrow T \| (enc, c)$
    return $c$

proc $\mathsf{Dec}_{\text{FL-}\varphi\text{CCA}}(c)$
    $m \leftarrow \mathsf{Dec}_{\text{FL-}\psi\text{CCA}}(c)$
    $p \leftarrow \varphi(T, c)$
    $T \leftarrow T \| (dec, c, p)$
    if $p \neq \lightning$
        $m \leftarrow \perp$
    return $m$

proc $\mathsf{Fin}_{\text{FL-}\varphi\text{CCA}}(\overline{d})$
    return $\mathsf{Fin}_{\text{LoR-(sf)AE}}(\overline{d})$
```

Note that because $\psi \subset \varphi$, any query for which $\varphi(T,c) = \lightning$ also satisfies $\psi(T,c) = \lightning$. So every query which should not be suppressed in the FL-φCCA game is also not suppressed in the FL-ψCCA game. So $\mathcal{A}_\psi$ simulates the FL-φCCA game with the same value of b and therefore has the same advantage. □

In order to prove a converse it would again be necessary to introduce some way of measuring whether an efficient adversary can cause a specific transcript to occur. The adversary needs an input where $\psi(T,c) \neq \lightning = \varphi(T,c)$, which would not be suppressed in the φ-FUL game, so an attack which uses the value of that decryption would work for φ but not for ψ.

Note that as the implications are reversed for privacy and correctness notions the games FL-φAE and FL-ψAE are in general incomparable for $\psi \subset \varphi$. This is a change from the AE game defined in Chapter 4 due to the fact that the new combined game also ensures correctness.

The next theorem shows that the relations between the different flavors of privacy are the same for every predicate.

Theorem 7.8
*Let φ be a predicate. The Theorems 2.6 (*RoR-CPA $\Rightarrow$ LoR-CPA*), 2.7 (*LoR-CPA $\Rightarrow$ RoR-CPA*), 2.8 (*LoR-CPA $\Rightarrow$ FtG-CPA*), 2.9 + 2.10 (*FtG-CPA $\rightarrow$ LoR-CPA*), 2.11 (*SEM-CPA $\Rightarrow$ FtG-CPA*), 2.12 (*FtG-CPA $\Rightarrow$ SEM-CPA*), 2.13 (*\$-CPA $\Rightarrow$ LoR-CPA*) and 2.14 (*LoR-CPA $\not\Rightarrow$ \$-CPA*) hold if we replace* CPA *with* φAE *or with* φCCA.

Also we have FtG-φAE $\rightarrow$ LoR-φCCA *and* LoR-φAE $\not\Rightarrow$ \$-φCCA. *If ψ is a second predicate so that $\psi \subset \varphi$, then we also have* FtG-ψCCA $\rightarrow$ LoR-φCCA *and*

$\mathsf{LoR}\text{-}\psi\mathsf{CCA} \not\Rightarrow \$\text{-}\varphi\mathsf{CCA}$.

Proof As with all iterations of this theorem so far, the decryption queries are the same every flavor, so the reductions (which only rely on the encryption and challenge queries) can be applied immediately.

The separation used in 2.17 to show $\mathsf{LoR}\text{-}\mathsf{CCA} \not\Rightarrow \$\text{-}\mathsf{CCA}$ also works for an arbitrary predicate. Indeed the scheme which adds and remove 0^n remains $\mathsf{LoR}\text{-}\varphi\mathsf{AE}$ secure for any predicate φ.

Similarly, the separation used in Theorem 4.8 to show that $\mathsf{FtG}\text{-}\mathsf{AE} \to \mathsf{LoR}\text{-}\mathsf{AE}$ also holds for an arbitrary predicate. □

We now show that combining predicates results in a predicate which ensures the fulfilment of both individual predicates. Among other things, this shows that a fully-specified predicate indeed is equivalent to the combination of privacy and fulfilment.

Theorem 7.9 (φ-FUL + FL-φCCA $\Leftrightarrow$ FL-φAE)
Let φ be a correctness notion. There exists a constant c so that for any k, t, q and μ we have

$$\begin{aligned}
\mathsf{Sec}^{\mathsf{FL}\text{-}\varphi\mathsf{CCA}}_{\mathcal{SE}}(k,t,q,\mu) &\leq \mathsf{Sec}^{\mathsf{FL}\text{-}\varphi\mathsf{AE}}_{\mathcal{SE}}(k,t+cq,q,\mu),\\
\mathsf{Sec}^{\varphi\text{-}\mathsf{FUL}}_{\mathcal{SE}}(k,t,q,\mu) &\leq 2\cdot \mathsf{Sec}^{\mathsf{FL}\text{-}\varphi\mathsf{AE}}_{\mathcal{SE}}(k,t+cq,q,\mu) \text{ and}\\
\mathsf{Sec}^{\mathsf{FL}\text{-}\varphi\mathsf{AE}}_{\mathcal{SE}}(k,t,q,\mu) &\leq \mathsf{Sec}^{\mathsf{FL}\text{-}\varphi\mathsf{CCA}}_{\mathcal{SE}}(k,t+cq,q,\mu) + \mathsf{Sec}^{\varphi\text{-}\mathsf{FUL}}_{\mathcal{SE}}(k,t,q,\mu).
\end{aligned}$$

Proof The first statement is fairly simple: we can simulate the FL-φCCA game by forwarding queries and always suppressing specified queries, independently of whether they are actually suppressed in the FL-φAE game.

For the second statement we use the same idea as in the proof of Theorem 4.13: Let $\mathcal{A}$ be an adversary for the φ-FUL game and consider a game $\mathsf{FL}^1\text{-}\varphi\mathsf{AE}$ where encryption queries are always answered as if $b = 1$ (and the other queries are normal). We define an adversary $\mathcal{A}'$ in Figure 7.5 which forwards the queries from $\mathcal{A}$, restores queries which return ε (as if they decrypted correctly, they can be restored) and finalizes with 1 if any specified decryption query returns something other than ε.

This adversary simulates the φ-FUL up until the first time $\mathcal{A}$ sends a query which would decrypt incorrectly. So it has the same advantage as $\mathcal{A}$. Then we can bound the advantage in the $\mathsf{FL}^1\text{-}\varphi\mathsf{AE}$ game by adding and subtracting the term $\Pr[\mathsf{FL}\text{-}\varphi\mathsf{AE}^{\mathcal{A}}_{\mathcal{SE}}(k,0) = 1]$. This gives one term which is exactly the definition advantage in the FL-φAE game and one term which is equivalent to the FL-φCCA (which we bounded above).

For the final statement we can proceed as in the proof of Theorem 4.12. Let $\mathcal{A}$ be an adversary for the FL-φAE game. Now add and subtract the term

```
$\mathcal{A}'$
1 :  proc $\mathsf{Init}_{\varphi\text{-FUL}}()$
2 :    $T \leftarrow \varnothing$
3 :    $E \leftarrow \varnothing$
4 :    $d \leftarrow 0$
5 :    return $\mathsf{Init}_{\mathsf{FL}^1\text{-}\varphi\mathsf{AE}}()(d)$

6 :  proc $\mathsf{Enc}_{\varphi\text{-FUL}}(m)$
7 :    $c \leftarrow \mathsf{Enc}_{\mathsf{FL}^1\text{-}\varphi\mathsf{AE}}(m)$
8 :    $T \leftarrow T \| (enc, c)$
9 :    $E \leftarrow E \| \{c : m\}$
10 :   return $c$

11 : proc $\mathsf{Dec}_{\varphi\text{-FUL}}(c)$
12 :   $m \leftarrow \mathsf{Dec}_{\mathsf{FL}^1\text{-}\varphi\mathsf{AE}}(c)$
13 :   $p \leftarrow \varphi(T, c)$
14 :   $T \leftarrow T \| (dec, c, p)$
15 :   if $p \neq E$ and $m \neq \varepsilon$
16 :     $d \leftarrow 1$
17 :   if $p = 0$ and $m = \varepsilon$
18 :     $m \leftarrow \perp$
19 :   if $p = 1$ and $m = \varepsilon$
20 :     $m \leftarrow E[c]$
21 :   return $m$

22 : proc $\mathsf{Fin}_{\varphi\text{-FUL}}()$
23 :   return $\mathsf{Fin}_{\mathsf{FL}^1\text{-}\varphi\mathsf{AE}}(d)$
```

Figure 7.5: The adversary used in the proof of Theorem 7.9.

$\Pr[\mathsf{FL}^1\text{-}\varphi\mathsf{AE}^{\mathcal{A}}_{\mathcal{SE}}(k,0) = 1]$ to the advantage of $\mathcal{A}$. This gives an advantage term for the FL^1-φAE game and an advantage for the FL-φCCA game. So we just need to construct an adversary for the φ-FUL game which simulates the FL^1-φAE game. We can do this by forwarding every query and suppressing specified queries. This is a correct simulates up until the first specified query which would not be suppressed, but if this occurs then the constructed adversary wins the φ-FUL game. □

As noted before, this theorem does not yet show that we can ensure correctness, integrity and privacy up to correctness with a single game, as a predicate which ensures integrity will suppress every query in the privacy game. The next theorem, generic composition for predicates, solves this problem.

Theorem 7.10 (Generic composition for predicates)
Let ψ be an integrity predicate and φ be another predicate which does not contradict ψ. There exists a constant c so that for any k, t, q and μ we have

$$\mathsf{Sec}^{\mathsf{FL}\text{-}\varphi\mathsf{CCA}}_{\mathcal{SE}}(k,t,q,\mu) \leq \mathsf{Sec}^{\mathsf{FL}\text{-}(\varphi\cup\psi)\mathsf{CCA}}_{\mathcal{SE}}(k,t,q,\mu) + 2 \cdot \mathsf{Sec}^{\psi\text{-}\mathsf{FUL}}_{\mathcal{SE}}(k,t+cq,q,\mu)$$

The intuition behind this theorem is fairly clear: for queries which fulfil integrity (so output $\perp$), it does not matter if we suppress them (which also sets them to $\perp$) in the privacy game. The traditional generic composition result follows from this: let φ be a fully-specified predicate and apply the theorem to φ^{cor} and φ^{int}. As FL-φCCA is equivalent to FL-CPA, this shows that

$$\mathsf{Sec}_{\mathcal{SE}}^{\mathsf{FL}\text{-}\varphi^{cor}\mathsf{CCA}}(k,t,q,\mu) \leq \mathsf{Sec}_{\mathcal{SE}}^{\mathsf{FL}\text{-}\mathsf{CPA}}(k,t,q,\mu) + 2\cdot \mathsf{Sec}_{\mathcal{SE}}^{\varphi^{int}\text{-}\mathsf{FUL}}(k,t+cq,q,\mu).$$

Proof This proof again follows the proof idea for generic composition from [BN00], but extends it to predicates. Let $\mathcal{A}$ be an adversary for the FL-φCCA game. We have

$$\begin{aligned}
&\mathsf{Adv}_{\mathcal{SE},\mathcal{A}}^{\mathsf{FL}\text{-}\varphi\mathsf{CCA}}(k) = \Pr[\mathsf{FL}\text{-}\varphi\mathsf{CCA}_{\mathcal{SE}}^{\mathcal{A}}(k,1)=1] - \Pr[\mathsf{FL}\text{-}\varphi\mathsf{CCA}_{\mathcal{SE}}^{\mathcal{A}}(k,0)=1] \\
&= \Pr[\mathsf{FL}\text{-}\varphi\mathsf{CCA}_{\mathcal{SE}}^{\mathcal{A}}(k,1)=1] - \Pr[\mathsf{FL}\text{-}(\varphi\cup\psi)\mathsf{CCA}_{\mathcal{SE}}^{\mathcal{A}}(k,1)=1] \qquad (7.1)\\
&+ \Pr[\mathsf{FL}\text{-}(\varphi\cup\psi)\mathsf{CCA}_{\mathcal{SE}}^{\mathcal{A}}(k,1)=1] - \Pr[\mathsf{FL}\text{-}(\varphi\cup\psi)\mathsf{CCA}_{\mathcal{SE}}^{\mathcal{A}}(k,0)=1] \qquad (7.2)\\
&+ \Pr[\mathsf{FL}\text{-}(\varphi\cup\psi)\mathsf{CCA}_{\mathcal{SE}}^{\mathcal{A}}(k,0)=1] - \Pr[\mathsf{FL}\text{-}\varphi\mathsf{CCA}_{\mathcal{SE}}^{\mathcal{A}}(k,0)=1] \qquad (7.3)
\end{aligned}$$

The term 7.2 is exactly the definition of $\mathsf{Adv}_{\mathcal{SE},\mathcal{A}}^{\mathsf{FL}\text{-}(\varphi\cup\psi)\mathsf{CCA}}(k)$. So it suffices to bound the terms 7.1 and 7.3 by $\mathsf{Sec}_{\mathcal{SE}}^{\psi\text{-}\mathsf{FUL}}(k,t,q,\mu)$. This means we need to construct an adversary for the ψ-FUL game which simulates a privacy game where the encryption is independent of b (either always 1 for 7.1 or 0 for 7.3) and either suppresses queries for φ or for $\varphi\cup\psi$. We saw in the proofs of Theorems 4.9 and 4.12 that we can simulate the encryption queries for $b=1$ or $b=0$ using a truthful encryption oracle for any flavor, so we omit stating this in detail. Instead consider an adversary which maintains a transcript, simulates encryption accordingly and handles decryption as follows:

```
proc $\mathsf{Dec}_{\mathsf{FL}\text{-}\varphi\mathsf{CCA}}(c)$
  $p \leftarrow (\varphi\cup\psi)(T,c)$
  $T \leftarrow T\|(dec,c,p)$
  $m \leftarrow \mathsf{Dec}_{\psi\text{-}\mathsf{FUL}}(c)$
  if $p \neq \lightning$
    $m \leftarrow \bot$
  return $m$
```

This adversary simulates the game up until the first query where a query would be suppressed by $\varphi\cup\psi$ but not by φ and the response to this query is not $\bot$. But if this occurs, then the adversary has won the ψ-FUL game, as it has sent a query specified by ψ to be $\bot$ which did not decrypt to $\bot$. So following the calculation from Theorem 4.12 gives us the desired result. □

Note that this proof only holds if ψ is an integrity predicate. Intuitively this makes sense: just because a scheme is correct we cannot stop suppressing queries which are bound by correctness in the privacy game. The proof above would fail when we try to bound against the fulfilment game with a correctness predicate. If an adversary sends a decryption query which is bound to be correct by $\varphi \cup \psi$ but not by ψ, it can tell which queries are being suppressed even if the scheme decrypts correctly, so the constructed adversary does not necessarily win the fulfilment game.

There is still a (fairly technical) step missing to argue that the combined game for a fully-specified predicate is equivalent to the combination of correctness, integrity and privacy. It is namely not yet clear that the fulfilment of a fully-specified predicate is equivalent to the fulfilment of the separate correctness and integrity predicates. One direction of the equivalence is given by Theorem 7.6, the other direction we prove now.

Theorem 7.11
Let φ and ψ be predicates which do not contradict each other. There for any k, t, q and μ we have

$$\mathsf{Sec}_{\mathcal{SE}}^{(\varphi\cup\psi)\text{-}\mathsf{FUL}}(k,t,q,\mu) \leq \mathsf{Sec}_{\mathcal{SE}}^{\varphi\text{-}\mathsf{FUL}}(k,t,q,\mu) + \mathsf{Sec}_{\mathcal{SE}}^{\psi\text{-}\mathsf{FUL}}(k,t,q,\mu)$$

Proof Let $\mathcal{A}$ be an adversary for the $(\varphi \cup \psi)$-FUL game. Consider the first decryption query from $\mathcal{A}$ so that *win* is set to 1. This decryption is specified either by φ or by ψ, so it would set *win* to 1 in one of the games. Let W be the event that this query is specified by φ and note that $\overline{W}$ implies that the query is specified by ψ. So we have

$$\begin{aligned}
&\mathsf{Adv}_{\mathcal{SE},\mathcal{A}}^{(\varphi\cup\psi)\text{-}\mathsf{FUL}}(k)\\
&\quad= \Pr[(\varphi\cup\psi)\text{-}\mathsf{FUL}_{\mathcal{SE}}^{\mathcal{A}}(k,1)=1 \mid W] + \Pr[(\varphi\cup\psi)\text{-}\mathsf{FUL}_{\mathcal{SE}}^{\mathcal{A}}(k,1)=1 \mid \overline{W}]\\
&\quad\quad- \Pr[(\varphi\cup\psi)\text{-}\mathsf{FUL}_{\mathcal{SE}}^{\mathcal{A}}(k,0)=1 \mid W] - \Pr[(\varphi\cup\psi)\text{-}\mathsf{FUL}_{\mathcal{SE}}^{\mathcal{A}}(k,0)=1 \mid \overline{W}]\\
&\quad\quad\quad= \mathsf{Adv}_{\mathcal{SE},\mathcal{A}}^{\varphi\text{-}\mathsf{FUL}}(k) + \mathsf{Adv}_{\mathcal{SE},\mathcal{A}}^{\psi\text{-}\mathsf{FUL}}(k),
\end{aligned}$$

as desired. □

This theorem has some interesting real-world applications. If we are trying to prove correctness and/or integrity for a scheme, it allows us to split the predicate up into arbitrary pieces and prove fulfilment for each piece individually. This could simplify the proof. It could also allow us to prove stronger bounds on some portions, giving insights into which parts of the predicate have a greater or lesser impact on bounds.

Note that this equivalence does not holds for the privacy notions. While one direction is given by Theorem 7.7, the other direction is clearly impossible, as it would show that CPA implies CCA security. Thankfully, due to generic composition it is enough to prove CPA-privacy in most scenarios.

Chapter 8

Progress hiding with predicates

We now wish to define a notion of progress-hiding based on predicates. Recall that in Definition 5.7 we defined a notion of progress-hiding for stateful security which ensured that the adversary could not win trivially by breaking sync in a progress query. Ideally we would like to define such a notion for an arbitrary predicate, which suppresses progress queries which would allow an adversary to win by using properties of the predicate. This can occur if any later output of the predicate differs depending on the value of b, so on whether progress queries occur or not. It is however not necessarily possible to efficiently determine which progress queries need to be suppressed for arbitrary predicates, as the worst-case scenario would require calculating every possible transcript.

So, instead of trying to suppress queries depending on a predicate φ, we define a notion of progress-hiding which depends not only on φ but also on a so-called suppression function $\mathcal{F}$, which takes as input a transcript T and outputs 1 if a progress query should be allowed and 0 it should be suppressed. We then define a notion which measures whether a suppression function successfully suppresses the correct queries.

Definition 8.1 (PH-$(\varphi, \mathcal{F})$CCA)
Let $\mathcal{F}$ be a suppression function and φ a predicate. The progress-hiding chosen-ciphertext security up to φ of an encryption scheme $\mathcal{SE}$ up to $\mathcal{F}$ with security parameter k against adversaries with runtime t using q queries totalling μ bits is defined by $\mathsf{Sec}^{\mathsf{PH}\text{-}(\varphi,\mathcal{F})\mathsf{CCA}}_{\mathcal{SE}}(k,t,q,\mu)$ using the game $\mathsf{PH}\text{-}(\varphi,\mathcal{F})\mathsf{CCA}_{\mathcal{SE}}(k,b)$ depicted in Figure 8.1. The input for encryption and progress queries are messages $m \in \mathcal{M}$, the input for the decryption queries are strings c and the input for the finalization query is a bit d.

Note that the variable *pre* ensures that we always allow progress queries if no encryption or decryption have taken place, so even if we take the suppression function which always outputs 0, we still cover the scenario where

Game $\mathsf{PH}\text{-}(\varphi,\mathcal{F})\mathsf{CCA}_{\mathcal{SE}}(k,b)$

```
proc $\mathsf{Init}_{\mathsf{PH}\text{-}(\varphi,\mathcal{F})\mathsf{CCA}}()$
  $K \leftarrow\!\$\, \mathsf{KGen}(k)$
  $T \leftarrow \varnothing$
  $pre \leftarrow 1$
  return $k$

proc $\mathsf{Prog}_{\mathsf{PH}\text{-}(\varphi,\mathcal{F})\mathsf{CCA}}(m)$
  if $b = 1$ and $(\mathcal{F}(T) = 1$ or $pre = 1)$
    $c \leftarrow \mathsf{Enc}_K(m)$
    $\mathsf{Dec}_K(c)$
    $T \leftarrow T \| (enc, c)$
    $T \leftarrow T \| (dec, c, \varphi(T, c))$
  return $\perp$

proc $\mathsf{Enc}_{\mathsf{PH}\text{-}(\varphi,\mathcal{F})\mathsf{CCA}}(m)$
  $pre \leftarrow 0$
  $c \leftarrow \mathsf{Enc}_K(m)$
  $T \leftarrow T \| (enc, c)$
  return $c$

proc $\mathsf{Dec}_{\mathsf{PH}\text{-}(\varphi,\mathcal{F})\mathsf{CCA}}(c)$
  $pre \leftarrow 0$
  $m \leftarrow \mathsf{Dec}_K(c)$
  $p \leftarrow \varphi(T, c)$
  $T \leftarrow T \| (dec, c, p)$
  return $m$

proc $\mathsf{Fin}_{\mathsf{PH}\text{-}(\varphi,\mathcal{F})\mathsf{CCA}}(d)$
  return $d$
```

Figure 8.1: The game for $\mathsf{PH}\text{-}(\varphi,\mathcal{F})\mathsf{CCA}$

an adversary misses the beginning of communication. This property cannot be expressed using only suppression functions, as the transcript does not contain the information whether queries were part of progress queries or part of encryption/decryption queries. Note that we can define predicates which cannot provide even this level of progress-hiding, for example a predicate which only allows performing a fixed number of encryptions. For such predicates we would have to suppress every progress query.

We now define a notion called suppression consistency which ensures that a suppression function suppresses the correct progress queries for a specific predicate. In the game for this notion two transcripts T_0 and T_1 are maintained. Both contain the encryption and decryption queries sent by an adversary. Progress queries however are only contained in T_1. An adversary wins if it can cause the output of φ or $\mathcal{F}$ to differ for the two transcripts. Intuitively, if an adversary cannot win this game, then $\mathcal{F}$ ensures that all of the queries we wish to suppress for φ are indeed suppressed.

Definition 8.2 (PH-(φ, $\mathcal{F}$)CON)
Let φ be a predicate and $\mathcal{F}$ be a suppressing function. The progress-hiding consistency of $\mathcal{F}$ for φ and an encryption scheme $\mathcal{SE}$ with security parameter k against adversaries with runtime t using q queries totalling μ bits is defined by $\mathsf{Sec}_{\mathcal{SE}}^{\mathsf{PH}\text{-}(\varphi,\mathcal{F})\mathsf{CON}}$ (k,t,q,μ) using the game $\mathsf{PH}\text{-}(\varphi,\mathcal{F})\mathsf{CON}_{\mathcal{SE}}(k,b)$ depicted in Figure 8.2. The input for encryption and progress queries are messages $m \in \mathcal{M}$ and the input for the

```
Game PH-$(\varphi,\mathcal{F})\mathsf{CON}_{\mathcal{SE}}(k)$

proc $\mathsf{Init}_{\text{PH-}(\varphi,\mathcal{F})\mathsf{CON}}()$
    $K \leftarrow\$ \mathsf{KGen}(k)$
    $T_0 \leftarrow \varnothing$
    $T_1 \leftarrow \varnothing$
    $win \leftarrow 0$
    $pre \leftarrow 1$
    return $k$

proc $\mathsf{Prog}_{\text{PH-}(\varphi,\mathcal{F})\mathsf{CON}}(m)$
    if $\mathcal{F}(T_0) = 1$ or $pre = 1$
        $c \leftarrow \mathsf{Enc}_K(m)$
        $T_1 \leftarrow T_1 \| (enc, c)$
        $T_1 \leftarrow T_1 \| (dec, c, \varphi(T_1, c))$
        if $\mathcal{F}(T_0) \neq \mathcal{F}(T_1)$
            $win \leftarrow 1$
    return $\perp$

proc $\mathsf{Enc}_{\text{PH-}(\varphi,\mathcal{F})\mathsf{CON}}(m)$
    $pre \leftarrow 0$
    $c \leftarrow \mathsf{Enc}_K(m)$
    $T_0 \leftarrow T_0 \| (enc, c)$
    $T_1 \leftarrow T_1 \| (enc, c)$
    if $\mathcal{F}(T_0) \neq \mathcal{F}(T_1)$
        $win \leftarrow 1$
    return $c$

proc $\mathsf{Dec}_{\text{PH-}(\varphi,\mathcal{F})\mathsf{CON}}(c)$
    $pre \leftarrow 0$
    $m \leftarrow \mathsf{Dec}_K(c)$
    $p_0 \leftarrow \varphi(T_0, c)$
    $p_1 \leftarrow \varphi(T_1, c)$
    $T_0 \leftarrow T_0 \| (dec, c, p_0)$
    $T_1 \leftarrow T_1 \| (dec, c, p_1)$
    if $p_0 \neq p_1$ or $\mathcal{F}(T_0) \neq \mathcal{F}(T_1)$
        $win \leftarrow 1$
    return $m$

proc $\mathsf{Fin}_{\text{PH-}(\varphi,\mathcal{F})\mathsf{CON}}()$
    return $win$
```

Figure 8.2: The game for PH-$(\varphi, \mathcal{F})$CON

decryption queries are strings c.

We can now prove a sufficient condition to achieve progress-hiding, namely that under the assumption that $\mathcal{F}$ correctly suppresses progress queries for φ, then the combination of PH-CPA and the fulfilment of φ imply progress-hiding up to $\mathcal{F}$.

Theorem 8.3
Let φ be a fully-specified predicate and $\mathcal{F}$ be a suppressing function. There exists a constant c so that for any k, t, q and μ we have

$$\begin{aligned}\mathsf{Sec}_{\mathcal{SE}}^{\text{PH-}(\varphi,\mathcal{F})\mathsf{CCA}}(k,t,q,\mu) &\leq \mathsf{Sec}_{\mathcal{SE}}^{\text{PH-CPA}}(k,t+cq,q,\mu)\\ &\quad + 2\cdot \mathsf{Sec}_{\mathcal{SE}}^{\varphi\text{-FUL}}(k,t+cq,q,\mu) + 2\cdot \mathsf{Sec}_{\mathcal{SE}}^{\text{PH-}(\varphi,\mathcal{F})\mathsf{CON}}(k,t+cq,q,\mu)\end{aligned}$$

We make two remarks before we prove this theorem. First of all note that \$-φAE $\Rightarrow$ PH-CPA + φ-FUL, so if we assume privacy, integrity and correctness for a predicate then to obtain progress-hiding it suffices to find an appropriate suppression function. Secondly, note that there is nothing forcing the suppression function to be minimal, e.g. to suppress as few queries as possible. Taking F to always be 0 gives a valid definition, but results in a weaker result. So when we consider suppression functions for concrete predicates, we ideally not only need to argue that the suppression function is consistent, but also that it suppresses as few queries as possible.

Proof This proof is similar to the proof of Theorem 5.10, but now has to account for general predicates and suppression functions. The main difficulty results from having to maintain the correct transcript, which can differ depending on progress queries. For arbitrary predicates, these differences in the transcript could cause φ or $\mathcal{F}$ to output different things, which makes it harder to jump to the games we wish to bound by. We proceed using a series of game hops: for $n \in \{1, \ldots 6\}$ consider the games G_n defined in Figure 8.3. All of these games share the same initialization, encryption and finalization queries:

```
1 : proc $\mathsf{Init}_{G_n}()$
2 :     $K \leftarrow\!\$\ \mathsf{KGen}(k)$
3 :     $T_0 \leftarrow \varnothing$
4 :     $T_1 \leftarrow \varnothing$
5 :     $E \leftarrow \varnothing$
6 :     $pre \leftarrow 1$
7 :     return $k$

8 : proc $\mathsf{Enc}_{G_n}(m)$
9 :     $pre \leftarrow 0$
10 :    $c \leftarrow \mathsf{Enc}_K(m)$
11 :    $T_0 \leftarrow T \| (enc, c)$
12 :    $T_1 \leftarrow T' \| (enc, c)$
13 :    $E \leftarrow \{c : m\}$
14 :    return $c$

15 : proc $\mathsf{Fin}_{G_n}(d)$
16 :    return $d$
```

Not every game needs every variable defined here, but for simplicity we state them here once with every variable.

We begin by transforming an adversary $\mathcal{A}$ for the PH-$(\varphi, \mathcal{F})$CCA game into one which distinguishes G_1 from G_6. Note that the only difference between these games is that in G_1 and G_6 some decryption queries are suppressed. Specifically, queries which return the value specified by φ applied to a transcript which does not contain progress queries are suppressed. Note that we again suppress queries with ε instead of $\perp$ so that we can distinguish queries which correctly decrypted from those which falsely returned $\perp$. We can construct an adversary which runs $\mathcal{A}$, forwards its queries and maintains a transcript (without progress queries). This adversary can then "restore" suppressed decryption queries to their correct value and thus correctly simulate the PH-$(\varphi, \mathcal{F})$CCA game.

G_1

```
 1 : proc $\mathsf{Prog}_{G_1}(m)$
 2 :
 3 :
 4 :
 5 :   return $\bot$
 6 : proc $\mathsf{Dec}_{G_1}(c)$
 7 :   $m \leftarrow \mathsf{Dec}_K(c)$
 8 :   $p \leftarrow \varphi(T_0, c)$
 9 :   $T_0 \leftarrow T_0 \| (dec, c, p)$
10 :   if $(p = 1$ and $m = E[c])$
11 :       or $(p = 0$ and $m = \bot)$
12 :     $m \leftarrow \varepsilon$
13 :   return $m$
```

G_2

```
 1 : proc $\mathsf{Prog}_{G_2}(m)$
 2 :
 3 :
 4 :
 5 :   return $\bot$
 6 : proc $\mathsf{Dec}_{G_2}(c)$
 7 :
 8 :
 9 :
10 :
11 :
12 :
13 :   return $\varepsilon$
```

G_3

```
 1 : proc $\mathsf{Prog}_{G_3}(m)$
 2 :   if $\mathcal{F}(T_0) = 1$ or $pre = 1$
 3 :     $c \leftarrow \mathsf{Enc}_K(m)$
 4 :     $\mathsf{Dec}_K(c)$
 5 :   return $\bot$
 6 : proc $\mathsf{Dec}_{G_3}(c)$
 7 :   $pre \leftarrow 0$
 8 :   $p \leftarrow \varphi(T_0, c)$
 9 :   $T_0 \leftarrow T_0 \| (dec, c, p)$
10 :
11 :
12 :
13 :   return $\varepsilon$
```

G_4

```
 1 : proc $\mathsf{Prog}_{G_4}(m)$
 2 :   if $\mathcal{F}(T_1) = 1$ or $pre = 1$
 3 :     $c \leftarrow \mathsf{Enc}_K(m)$
 4 :     $p \leftarrow \varphi(T_1, c)$
 5 :     $\mathsf{Dec}_K(c)$
 6 :     $E \leftarrow E \| \{c : m\}$
 7 :     $T_1 \leftarrow T_1 \| (enc, c)$
 8 :     $T_1 \leftarrow T_1 \| (dec, c, p)$
 9 :   return $\bot$
10 : proc $\mathsf{Dec}_{G_4}(c)$
11 :   $pre \leftarrow 0$
12 :
13 :
14 :
15 :   $p_1 \leftarrow \varphi(T_1, c)$
16 :   $T_1 \leftarrow T_1 \| (dec, c, p_1)$
17 :
18 :
19 :
20 :   return $\varepsilon$
```

G_5

```
 1 : proc $\mathsf{Prog}_{G_5}(m)$
 2 :   if $\mathcal{F}(T_1) = 1$ or $pre = 1$
 3 :     $c \leftarrow \mathsf{Enc}_K(m)$
 4 :     $p \leftarrow \varphi(T_1, c)$
 5 :     $\mathsf{Dec}_K(c)$
 6 :     $E \leftarrow E \| \{c : m\}$
 7 :     $T_1 \leftarrow T_1 \| (enc, c)$
 8 :     $T_1 \leftarrow T_1 \| (dec, c, p)$
 9 :   return $\bot$
10 : proc $\mathsf{Dec}_{G_5}(c)$
11 :   $pre \leftarrow 0$
12 :   $m \leftarrow \mathsf{Dec}_K(c)$
13 :
14 :
15 :   $p_1 \leftarrow \varphi(T_1, c)$
16 :   $T_1 \leftarrow T_1 \| (dec, c, p_1)$
17 :   if $(p_1 = 1$ and $m = E[c])$
18 :       or $(p_1 = 0$ and $m = \bot)$
19 :     $m \leftarrow \varepsilon$
20 :   return $m$
```

G_6

```
 1 : proc $\mathsf{Prog}_{G_6}(m)$
 2 :   if $\mathcal{F}(T_1) = 1$ or $pre = 1$
 3 :     $c \leftarrow \mathsf{Enc}_K(m)$
 4 :     $p \leftarrow \varphi(T_1, c)$
 5 :     $\mathsf{Dec}_K(c)$
 6 :     $E \leftarrow E \| \{c : m\}$
 7 :     $T_1 \leftarrow T_1 \| (enc, c)$
 8 :     $T_1 \leftarrow T_1 \| (dec, c, p)$
 9 :   return $\bot$
10 : proc $\mathsf{Dec}_{G_6}(c)$
11 :   $pre \leftarrow 0$
12 :   $m \leftarrow \mathsf{Dec}_K(c)$
13 :   $p_0 \leftarrow \varphi(T_0, c)$
14 :   $T_0 \leftarrow T_0 \| (dec, c, p_0)$
15 :   $p_1 \leftarrow \varphi(T_1, c)$
16 :   $T_1 \leftarrow T_1 \| (dec, c, p_1)$
17 :   if $(p_0 = 1$ and $m = E[c])$
18 :       or $(p_0 = 0$ and $m = \bot)$
19 :     $m \leftarrow \varepsilon$
20 :   return $m$
```

Figure 8.3: The games used in the proof of Theorem 8.3.

Next we show that for any adversary $\mathcal{A}$ the following statements hold:

$$\Pr[G_2{}^{\mathcal{A}}_{\mathcal{SE}} = 1] - \Pr[G_1{}^{\mathcal{A}}_{\mathcal{SE}} = 1] \le \mathsf{Sec}^{\varphi\text{-}\mathsf{FUL}}_{\mathcal{SE}}(k, t + cq, q, \mu) \tag{8.1}$$
$$\Pr[G_3{}^{\mathcal{A}}_{\mathcal{SE}} = 1] - \Pr[G_2{}^{\mathcal{A}}_{\mathcal{SE}} = 1] \le \mathsf{Sec}^{\mathsf{PH\text{-}CPA}}_{\mathcal{SE}}(k, t + cq, q, \mu) \tag{8.2}$$
$$\Pr[G_4{}^{\mathcal{A}}_{\mathcal{SE}} = 1] - \Pr[G_3{}^{\mathcal{A}}_{\mathcal{SE}} = 1] \le \mathsf{Sec}^{\mathsf{PH\text{-}}(\varphi,\mathcal{F})\mathsf{CON}}_{\mathcal{SE}}(k, t + cq, q, \mu) \tag{8.3}$$
$$\Pr[G_5{}^{\mathcal{A}}_{\mathcal{SE}} = 1] - \Pr[G_4{}^{\mathcal{A}}_{\mathcal{SE}} = 1] \le \mathsf{Sec}^{\varphi\text{-}\mathsf{FUL}}_{\mathcal{SE}}(k, t + cq, q, \mu) \tag{8.4}$$
$$\Pr[G_6{}^{\mathcal{A}}_{\mathcal{SE}} = 1] - \Pr[G_5{}^{\mathcal{A}}_{\mathcal{SE}} = 1] \le \mathsf{Sec}^{\mathsf{PH\text{-}}(\varphi,\mathcal{F})\mathsf{CON}}_{\mathcal{SE}}(k, t + cq, q, \mu) \tag{8.5}$$

Consider first (8.1). In both G_1 and G_2, progress queries are suppressed. For G_2 all decryption queries are suppressed and for G_1 only queries which decrypt correctly according to φ. Intuitively, an adversary which can distinguish these games can make the scheme decrypt a ciphertext incorrectly. Formally, we can construct an adversary which forwards encryption and decryption queries to the φ-FUL game and maintains a transcript as input for φ. Progress queries are always answered with $\perp$, decryption queries with ε. This simulates the games G_2 and simulates the games G_1 up until the first query which would decrypt incorrectly. If this occurs, then the constructed adversary wins the φ-FUL game.

The argument for (8.4) is almost the same. The only difference is that now progress queries are allowed. But we can simulate progress queries with an encryption and decryption query in the φ-FUL game. Note this allows us to maintain the transcript correctly, as we see the ciphertext in the simulated progress query. The rest of the argument is the same.

Next consider (8.2). In G_2 and G_3, all decryption queries are suppressed. In G_2 progress queries are also suppressed. In G_3 they are not, but the progress queries are never added to the transcript. So an adversary which can distinguish this game can tell whether progress queries are occurring. This is almost the PH-CPA game, with the small difference that some progress queries are suppressed by $\mathcal{F}$. We can however construct an adversary for the PH-CPA game which maintains a transcript and forwards encryption queries. Decryption queries are added to the transcript, but otherwise ignored. Progress queries are also forwarded, unless they are prohibited by $\mathcal{F}$. Note that it is important that the transcript does not contain progress queries, as otherwise the constructed adversary could not maintain the correct transcript. Since this is the case in G_3, this simulation is correct.

Now we argue why (8.3) holds. In G_3 and G_4, decryption queries are still suppressed. Progress queries always occur, but in G_4 they are included in the transcript, in G_3 they are not. These games return identical outputs unless the adversary can send a progress query for which the value of $\mathcal{F}$ differs depending on whether previous progress queries were included in the transcript. Denote such a query a distinguishing query. A distinguishing query will cause a progress query to occur in one game, but not in the other. This does not immediately allow the adversary to win the game, but it might cause encryption queries to output different values. However, if we forward every query of the adversary to the PH-$(\varphi, \mathcal{F})$CON game, then we simulate G_3 and G_4 at least until a distinguishing query occurs and if such a query occurs, then we win the consistency game. So the stated bound holds.

Finally, consider (8.5). In G_5 and G_6, progress queries are allowed and contained in the transcript. Decryption queries are suppressed if the decryption is correct. In G_5 we check this using the transcript including the progress queries. In G_6 we maintain a separate transcript which does not include progress queries. So an adversary which can distinguish these games can send a query which causes φ to output something different depending on whether progress queries are contained in the transcript. So consider an adversary which maintains a transcript (which cannot contain progress queries) and forwards queries to the PH-$(\varphi, \mathcal{F})$CON game, suppressing decryption queries which decrypted correctly. This simulates G_6 and simulates G_5 up until the first distinguishing query, which causes a victory in the consistency game. $\square$

The assumption that φ is fully-specified is not necessary for most of this proof, in most of the steps one could suppress or check specified queries instead of all queries without any issue. The one step where we have an issue is where we try to construct an adversary for the PH-CPA game. Here it is necessary that all decryption queries are suppressed. One could define a notion of "progress-hiding up to correctness" where specified queries are suppressed instead of all queries, being careful to still account for suppression functions. Then one could presumably prove this theorem for general predicates. Since we argued in the last chapter that the practical use of non-fully-specified predicates is fairly limited and this proof is already fairly complex, we refrain from doing this.

8.1 Suppressing functions for common predicates

We now take a look what progress-hiding means for the concrete correctness predicates defined in the previous chapters. By Theorem 8.3, this means finding an appropriate suppression function, so an $\mathcal{F}$ so that PH-$(\varphi, \mathcal{F})$CON is small but $\mathcal{F}$ suppresses as few queries as possible. We will see that the most minimal suppression function F_1, which always outputs 1, works for some, but not all of the common predicates.

Replay protection

We begin with the replay-protection predicate φ_{rep}. Here F_1 can be used: clearly $\mathsf{F}_1(T_0) = 1 = \mathsf{F}_1(T_1)$ always holds. Now consider $\varphi_{rep}(T_0, c)$ and

$\varphi_{rep}(T_1, c)$. The output of these are independent of other ciphertexts, so they will always be equal for ciphertexts which we not generated in progress queries. For ciphertexts generated in progress queries, φ_{rep} will always return 0, either because the ciphertext has not been encrypted or because it has already been decrypted. So we have $\mathsf{PH}\text{-}(\varphi_{rep}, \mathsf{F}_1)\mathsf{CON} = 0$.

Stateless correctness and dropping protection

Now let us consider the stateless predicate φ_{sl}. Recalling PH-CCA, one would expect that no queries need to be suppressed. However, unlike for the replay predicate, for ciphertexts generated in progress queries we have $\varphi_{sl}(T_0, c) = 0 \neq 1 = \varphi_{sl}(T_1, c)$. So if an adversary could guess this ciphertext, it would win the consistency game. If the scheme provides privacy, we would expect the probability of guessing this correctly to be equal to the probability of guessing a random string. We prove this in the following theorem.

Theorem 8.4
Let F_1 be the suppressing function which always outputs 1*. Let $c_{min} = \min\{|c| \mid c \in \mathcal{C}\}$ and fix q. There exists a constant d so that for any k, t and μ we have*

$$\mathsf{Sec}_{\mathcal{SE}}^{\mathsf{PH}\text{-}(\varphi_{sl},\mathsf{F}_1)\mathsf{CON}}(k,t,q,\mu) \leq \mathsf{Sec}_{\mathcal{SE}}^{\$\text{-}\varphi_{sl}\mathsf{AE}}(k,t+dq,q,\mu) + \frac{q^2}{2^{c_{min}+2}}.$$

Proof Let $\mathcal{A}$ be an adversary for the $\mathsf{PH}\text{-}(\varphi_{sl}, \mathsf{F}_1)\mathsf{CON}$ game. Note that we always have $F_1(T_0) = 1 = F_1(T_1)$, so the only way this adversary can win is if $\varphi_{sl}(T_0, c) \neq \varphi_{sl}(T_1, c)$ for some c. By the definition of φ_{sl}, the only ciphertexts for which this can happen are the ciphertexts generated in a progress query.

So consider the adversary $\mathcal{A}_\$$ for the $\$\text{-}\varphi_{sl}\mathsf{AE}$ game defined in Figure 8.4. Encryption queries are forwarded and saved so that decryption queries can be restored. Progress queries are encrypted and the resulting ciphertexts are saved. Decryption queries are forwarded and restored if they are suppressed by the $\mathsf{PH}\text{-}(\varphi_{sl}, \mathsf{F}_1)\mathsf{CON}$ game. If $\mathcal{A}_\$$ ever sends one of the ciphertexts from a progress query in a decryption query, then $\mathcal{A}_\$$ finalizes with 1, otherwise with 0.

Consider first $b = 1$ in the $\$\text{-}\varphi_{sl}\mathsf{AE}$ game. This game only suppresses decryption queries if they decrypt correctly, so $\mathcal{A}_\$$ restores them to the correctly and therefore simulates the $\mathsf{PH}\text{-}(\varphi_{sl}, \mathsf{F}_1)\mathsf{CON}$. Now consider a decryption query where $\mathcal{A}$ sends a decryption query with ciphertext c which was generated in a progress query. As $c \in P$, $\mathcal{A}_\$$ will output 1. As this is the only way an adversary can win the $\mathsf{PH}\text{-}(\varphi_{sl}, \mathsf{F}_1)\mathsf{CON}$ game we have

$$\Pr[\mathsf{PH}\text{-}(\varphi_{sl}, \mathsf{F}_1)\mathsf{CON}_{\mathcal{SE}}^{\mathcal{A}}(k) = 1] = \Pr[\$\text{-}\varphi_{sl}\mathsf{AE}_{\mathcal{SE}}^{\mathcal{A}_\$}(k, 1) = 1].$$

For $b = 0$ the strings in P were sampled randomly in the $\$\text{-}\varphi_{sl}\mathsf{AE}$ game. The only way that $\mathcal{A}_\$$ will output 1 is if one of these random strings generated

```
$\mathcal{A}_\$$

proc $\mathsf{Init}_{\text{PH-}\varphi_{sl}\text{SUP}}()$
    $P \leftarrow \varnothing$
    $S \leftarrow \varnothing$
    $E \leftarrow \varnothing$
    $d \leftarrow 0$
    return $\mathsf{Init}_{\$\text{-}\varphi_{sl}\text{AE}}()$

proc $\mathsf{Prog}_{\text{PH-}\varphi_{sl}\text{SUP}}(m)$
    $c \leftarrow \mathsf{Enc}_{\$\text{-}\varphi_{sl}\text{AE}}(m)$
    $P \leftarrow P \cup c$
    return $\perp$

proc $\mathsf{Enc}_{\text{PH-}\varphi_{sl}\text{SUP}}(m)$
    $c \leftarrow \mathsf{Enc}_{\$\text{-}\varphi_{sl}\text{AE}}(m)$
    $S \leftarrow S \cup \{c\}$
    $E \leftarrow E \cup \{c : m\}$
    return $c$

proc $\mathsf{Dec}_{\text{PH-}\varphi_{sl}\text{SUP}}(c)$
    if $c \in P$
        $d \leftarrow 1$
    $m \leftarrow \mathsf{Dec}_{\$\text{-}\varphi_{sl}\text{AE}}(m)$
    if $m = \varepsilon$
        $m \leftarrow \perp$
        if $c \in S$
            $m \leftarrow E[c]$
    return $m$

proc $\mathsf{Fin}_{\text{PH-}phi_{sl}\text{SUP}^{\mathcal{F}}}()$
    return $\mathsf{Fin}_{\$\text{-}\varphi_{sl}\text{AE}}(d)$
```

Figure 8.4: The adversary used in the proof of Theorem 8.4.

in a progress query is equal to one of the ciphertexts sent in a decryption query. There are at most $\frac{q^2}{4}$ pairs of progress query and decryption query and each pair has a probability of at most $2^{-c_{min}}$ of being equal. So the stated bound holds. □

Similarly, for the dropping predicate φ_{drop} progress queries make no difference for future queries. As the progress query performs both an encryption and a decryption, it has no impact on the question whether all previous ciphertexts have been decrypted. The only exception are again ciphertexts generated in progress queries, so using the same argument above we can bound PH-$(\varphi_{drop}, \mathsf{F}_1)$CON.

Reorder protection

For the reorder predicate φ_{ord} we need a more complex suppression function. Let F_{latest} be the suppression function which allows progress queries as long as the encryption and decryption algorithms are at the same point in the sequence of messages. Formally, define F_{latest} as

$$F_{latest}(T) = \begin{cases} 1 & \textbf{if } latest(T) = |T_{enc}| - 1 \\ 0 & \textbf{else} \end{cases}$$

We first argue why this suppression function is indeed minimal. Assume that we have a suppression function F which allows a progress query when the last encrypted ciphertext has not been decrypted. Now consider an adversary which causes this progress query to occur and then sends a decryption query with one of the previous ciphertexts which have not been decrypted. If the progress query occurred, then the decryption is out-of-order, otherwise it is not, so φ_{rep} will output different values. So this adversary would always win the PH-$(\varphi_{rep}, \mathsf{F})$CON game.

Note that this example gives us a real-world progress attack that will always succeed against a scheme with reorder protection: if an adversary knows when it will lose access to the channel, it can suppress the last message beforehand, and upon returning send this message. If progress occurred, then the message will be rejected, otherwise it will be accepted. Theorem 8.3 tells us that if our scheme achieves \$-φ_{ord}AE, then this is the only type of attack that can succeed.

Now we argue that this suppression function is indeed sufficient, so we wish to bound $\mathsf{Sec}_{\mathcal{SE}}^{\mathsf{PH}\text{-}(\varphi_{ord},\mathsf{F}_{\mathsf{latest}})\mathsf{CON}}(k,t,q,\mu)$. We will actually argue that it is impossible for an adversary to cause φ_{ord} or F_{latest} to output different values using an inductive argument. We consider each type of query and show that if $F_{latest}(T_0) = F_{latest}(T_1)$ before a query, then this also holds after the query. For decryption queries, we also need argue that $\varphi_{ord}(T_0, c) = \varphi_{ord}(T_1, c)$.

We begin with progress queries. If a progress query is suppressed, then clearly no change occurs. If it is not suppressed, then $F_{latest}(T_0) = F_{latest}(T_1) = 1$ before the query. Adding an encryption and a decryption to T_1 does not change this, as the ciphertext encrypted becomes latest ciphertext and it is decrypted.

For encryption queries note that independent of the values before the query, afterwards we have $F_{latest}(T_0) = F_{latest}(T_1) = 0$, as a new ciphertext was encrypted but not decrypted yet.

For decryption queries consider first if $F_{latest}(T_0) = F_{latest}(T_1) = 1$. Then adding a further decryption cannot undo the decryption of the latest ciphertext. So afterwards we still have $F_{latest}(T_0) = F_{latest}(T_1) = 1$. Also, the ciphertext c sent in the decryption query cannot have a greater index than $latest(T)$, so we have $\varphi_{ord}(T_0, c) = \varphi_{ord}(T_1, c) = 0$.

On the other hand, if $F_{latest}(T_0) = F_{latest}(T_1) = 0$ then the latest ciphertext c' was not yet decrypted. If $c = c'$, then $F_{latest}(T_0) = F_{latest}(T_1) = 1$ afterwards, otherwise $F_{latest}(T_0) = F_{latest}(T_1) = 0$. Finally, consider the output of φ_{rep} and note that the encryption of c' must have occurred in an encryption query, not a progress query. So for every ciphertext not generated in a progress query, $ind_{enc}(c) > latest(T)$ will evaluate to the same value. For ciphertext generated in progress queries, φ_{ord} will output 0 for both transcripts, as for T_0 this ciphertext is a forgery and for T_1 its index is not increasing.

So all together we have $\mathsf{Sec}_{\mathcal{SE}}^{\mathsf{PH}\text{-}(\varphi_{ord},\mathsf{F}_{\mathsf{latest}})\mathsf{CON}}(k,t,q,\mu) = 0$.

One could also define a "strict" version of $F_{latest}(T)$ which allows all progress queries once a ciphertext has been rejected. The progress-hiding game for this function and the predicate φ_{sf} is indeed equivalent to the original progress-hiding notion PH-sfCCA.

Acceptance windows

Finally, let us consider the acceptance-window predicate $\varphi_{(w_p,w_f)}$ with $w_p > 0$. This predicate requires a fairly strict suppression function. We begin with some examples to illustrate this.

Consider an adversary which encrypts a message and then sends w_p progress queries. Depending on whether these occur, the first ciphertext c_1 is either still in the acceptance-window or just outside of it, resulting in a different output from $\varphi_{(w_p,w_f)}$. So after w_p messages have been encrypted, every progress query has to be suppressed. Note that we could also replace all but the last progress queries with encryption queries.

Similarly, consider an adversary which encrypts $w_f + 1$ messages and sends a progress query. The last ciphertext c_{w_f+1} is either just outside of the future window or contained in the past window, again resulting in different outputs for $\varphi_{(w_p,w_f)}$.

So simply the existence of an acceptance-window makes it possible for an adversary to tell if progress occurred by testing whether ciphertexts are still contained in the window. A consistent suppression function can therefore only allow progress queries until $\min\{w_p, w_f\}$ encryption have occurred. If w_p and w_f are functions then the minimum needs to taken over the possible outputs. Specifically, let F_{w_p,w_f} be defined by

$$F_{w_p,w_f} = \begin{cases} 1 & \textbf{if } |T_{enc}| < \min\{w_p, w_f\} \\ 0 & \textbf{else} \end{cases}$$

By the same argument as above, we can argue that the chance of guessing a ciphertext from a progress query is small. For all other ciphertexts, note that they are guaranteed to be contained in the acceptance window independently of whether progress queries occur. So this function is sufficient.

This suppression function is not quite minimal. One could ignore progress queries which are sent while $pre = 1$ when determining the length of the transcript. As it is not possible to tell from the transcript whether a query was a progress query or a encryption/decryption query, the syntax presented here cannot express this. One could presumably fix this by defining the suppression function to take as input a transcript which takes as input the queries of the progress hiding game instead of the transcript of the encryption scheme. If w_p and w_f are functions one could also try to dynamically change the number depending on the ciphertexts sent. This is also somewhat tedious, so we omit it and instead just present this simpler, but not perfectly minimal suppression function.

Chapter 9

Analysis of DTLS

The Datagram Transport Layer Security Protocol (DTLS) is based on the TLS protocol, but is meant to used on unreliable network protocols, like UDP. This makes it impossible to provide complete replay and reorder protection. It also makes it impossible to use implicit nonces, which allowed TLS 1.3 to achieve \$-sfAE. To solve this, DTLS 1.3 uses a construction based on [BNT19] where the nonce is XORed with a mask generated by applying a pseudo-random function (PRF) to a portion of the ciphertext. As we see in this chapter, this allows DTLS 1.3 to achieve privacy using \$-type oracles.

Before analysing DTLS, we need to define what a PRF is and what it means for one to be secure. Consider a function which maps from a keyspace $\mathcal{K}_{\mathsf{PRF}}$ and a domain $\mathcal{D}$ to a range $\mathcal{R}$. The pseudo-random security of such a function is defined as the advantage of an adversary trying to distinguish such a function for a fixed but secret key from a function sampled randomly from the set $Fun(\mathcal{D},\mathcal{R})$ of all function from $\mathcal{D}$ to $\mathcal{R}$.

```
PRF-SEC_F(k, b)

1 : proc Init_PRF-SEC()
2 :     $K \leftarrow\$ mathcalK_{\mathsf{PRF}}$
3 :     $\tilde{\mathsf{F}} \leftarrow\$ Fun(\mathcal{D},\mathcal{R})$
4 :     return $k$

7 : proc Eval_PRF-SEC($x$)
8 :     $y_0 \leftarrow \tilde{\mathsf{F}}(x)$
9 :     $y_1 \leftarrow \mathsf{F}(K, x)$
10 :    return $y_b$

11 : proc Fin_PRF-SEC($d$)
12 :     return $d$
```

We also need to slightly extend some notions for the underlying scheme to allow for nonces and additional data as we did in Chapter 6. Instead of just a combined notion, we adjust the two notions φ_{sl}-FUL and \$-CPA. The adjustments are similar as for \$-AE: The encryption and decryption queries take nonce and additional data as extra inputs and nonces can only be used once for encrypting. The transcript to which we apply φ_{sl} includes the nonce and additional data, so $\varphi_{sl}(T,(n,c,ad)) = 1$ if $(enc,(n,c,ad)) \in T$ and 0 otherwise.

Unlike for TLS, we also need to adjust the notions we wish to apply to DTLS itself. This is because a dynamic acceptance window can be defined for every ciphertext individually. The size of this window ws is therefore an additional input for the encryption algorithm. We allow adversaries to choose this size for each encryption queries.

Next, let us define the correctness predicate φ_{dtls} for DTLS. It combines an acceptance window and replay protection. Ciphertext are only accepted in a window of size 2^8 or 2^{16} around the latest decrypted ciphertext. This size is determined dynamically for each ciphertext. Additionally, there is a replay-window of size w_r. Ciphertexts inside this window are checked for replays and ciphertexts outside of this window are rejected. So the past boundary of the acceptance window is in fact the minimum of w_r and the value given by the ciphertext. Formally, let $ws(c) \in \{8,16\}$ be the bitlength of the dynamic acceptance window given by a ciphertext c and define $(w_p(c) = \min\{w_r, 2^{ws(c)-1}\}$ and $w_f(c) = 2^{ws(c)-1}$ and define

$$\varphi_{DTLS}(T,c) = \begin{cases} 1 & \textbf{if } (enc,c) \in T \text{ and } (dec,c,1) \notin T \\ & \text{and } -w_p(c) \leq ind_{enc}(c) - latest(T) \leq w_f(c) \\ 0 & \textbf{else} \end{cases}$$

As for TLS, we analyse an abstract version of the DTLS 1.3 protocol, shown in Figure 9.1. This abstraction mostly involves ignoring header fields and replacing the underlying primitives with idealized versions. A DTLS 1.3 headers consists of the following fields:

- An optional connection ID, which can be used when it is hard to identify the target of the record.
- An optional length field, which is used if multiple records are sent in a single datagram.
- An epoch number, which starts at 0 and increases whenever keys are renegotiated.
- A (masked) partial sequence number which is either 8 or 16 bits long.
- A series of bits which indicate whether the optional fields are present and what the length of the sequence number is.

We assume the first two fields (and their bit indicators) are a part of the transmission and therefore ignore them. We also only consider the scenario for a single set of keys, so we ignore the epoch number. We also ignore the bit which indicates the length of the partial sequence number. Similarly to

TLS, DTLS does not attempt to hide length information and this extends to the length of the sequence number. So it is clearly impossible for DTLS to achieve \$-privacy if we include this bit.

Note the protocol QUIC, which in many aspects is very similar to DTLS, masks not only the sequence number but also the bit which indicates its length. A formal analysis of this construction can be found in [DFP$^+$20]. The authors there show that QUIC ciphertexts including the protection of the length of the sequence number achieve privacy using \$-type encryption oracles. It is difficult to express this construction with our predicates, as if the length of the dynamic acceptance window is masked, the predicate would require the PRF key to parse the size of the window. One could presumably extend the transcript to include additional information, specifically one could add the window size to each encryption entry. This is outside the scope of this analysis.

So the ciphertext we are analysing consists of the partial sequence number and the AEAD output, which we assume to be encoded in a vector. We also assume that this encoding does not allow an adversary to distinguish the ciphertext from random strings. In the actual DTLS protocol, the encoding is simply the concatenation of two values and the single bit indicating the length of the first. While this bit does allow an adversary to distinguish from random strings, all it really leaks it the length of the partial sequence number, which we are not trying to hide. So while the assumption that the encoding looks perfectly random is not quite correct, the information gained through this mistake is explicitly omitted from this analysis.

The encryption of a message is almost the same as for TLS 1.3. The sequence number is XORed with a fixed value agreed upon during the key exchange to produce a nonce. The message is than encrypted using the AEAD scheme, using the nonce and additional data consisting of the header, which in this case is only the (unmasked) 8 or 16 least significant bits of the sequence number. Finally, we mask the partial sequence number by applying a PRF to the first 16 bytes of the AEAD output and XORing its leading bits with the partial sequence number. Note that this means that the ciphertext must be at least 16 bytes long. The final output is the masked segment and the AEAD output.

Decryption reverses this process: first the masked segment is unmasked by applying the PRF to the ciphertext. Then the sequence number is restored to be the value which is closest to the most recently received ciphertext sequence number *latest* and whose lowest bits correspond to the unmasked segment. We denote this operation by $closest(latest, seq_{partial})$. Then the nonce can be calculated and the AEAD decryption can be performed. If the AEAD scheme rejects the decryption or the sequence number is outside of the replay window w_r or the sequence number is repeated, then the de-

Scheme $\mathcal{SE}$

```
proc $\mathsf{KGen}_{\mathcal{SE}}(k)$
  $K_{\mathsf{AEAD}} \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{AEAD}}(k)$
  $K_{\mathsf{F}} \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{F}}(k)$
  $\kappa \leftarrow\!\!\$\ \{0,1\}^N$
  $seq \leftarrow 0$
  $latest \leftarrow 0$
  $S \leftarrow \varnothing$
  return $((K_{\mathsf{AEAD}}, K_{\mathsf{F}}, \kappa), seq, (latest, S))$

proc $\mathsf{Enc}_{\mathcal{SE}}((K_{\mathsf{AEAD}}, K_{\mathsf{F}}, \kappa), m, (seq, ws))$
  if $ws \notin \{8,16\}$ or $l(|m|) < 128$
    return $(\perp, seq)$
  else
    $seq \leftarrow seq + 1$
    $n \leftarrow \kappa \oplus seq$
    $seq_{partial} \leftarrow seq[N - ws + 1, N]$
    $c \leftarrow \mathsf{Enc}_{\mathsf{AEAD}}(K_{\mathsf{AEAD}}, n, m, seq_{partial})$
    $mask \leftarrow \mathsf{F}(K_{\mathsf{F}}, c[0,127])$
    $\tilde{n} \leftarrow seq_{partial} \oplus mask[0, ws - 1]$
    return $((\tilde{n}, c), seq)$

proc $\mathsf{Dec}_{\mathcal{SE}}((K_{\mathsf{AEAD}}, K_{\mathsf{F}}, \kappa), (\tilde{n}, c), (latest, S))$
  $mask \leftarrow \mathsf{F}(K_{\mathsf{F}}, c[0,127])$
  $seq_{partial} \leftarrow \tilde{n} \oplus mask[0, |\tilde{n}| - 1]$
  $seq \leftarrow closest(latest, seq_{partial})$
  $n \leftarrow \kappa \oplus seq$
  $m \leftarrow \mathsf{Dec}_{\mathsf{AEAD}}(K_{\mathsf{AEAD}}, n, c, seq_{partial})$
  if $latest - seq > w_r$ or $seq \in S$
    $m \leftarrow \perp$
  if $m \neq \perp$
    $S \leftarrow S \cup seq$
    if $seq > latest$
      $latest \leftarrow seq$
  return $(m, (latest, S))$
```

Figure 9.1: The abstract scheme representing DTLS 1.3 with a replay window of size w_r.

cryption is rejected. Otherwise the received sequence numbers and *latest* is updated and the AEAD decryption is returned. Note that our abstraction saves all of the received indexes for the replay protection, while DTLS only saves those in the replay window. This is of course more space-efficient in real-world application, but makes no difference from a security perspective, as all of ciphertexts outside of this window are rejected anyway.

Theorem 9.1
Let $\mathcal{SE}$ be the scheme described above with a window size of w_r. There exists a constant c so that for any k, t, q and μ we have

$$\mathsf{Sec}_{\mathcal{SE}}^{\$\text{-}\varphi_{DTLS}\mathsf{AE}}(k,t,q,\mu) \leq \mathsf{Sec}_{\mathsf{AEAD}}^{\varphi_{sl}\text{-}\mathsf{FUL}}(k,t+cq,q,\mu) + \mathsf{Sec}_{\mathcal{SE}}^{\$\text{-}\mathsf{CPA}}(k,t,q+cq,\mu) + \mathsf{Sec}_{F}^{\mathsf{PRF\text{-}SEC}}(k,t,q,\mu) + \frac{q(q-1)}{2^{128}}.$$

Proof Note that because φ_{DTLS} is fully-specified, the \$-φ_{DTLS}CCA game is equivalent to the \$-CPA game. So by Theorem 7.9 we have for a constant $\tilde{c}$

$$\mathsf{Sec}_{\mathcal{SE}}^{\$\text{-}\varphi_{DTLS}\mathsf{AE}}(k,t,q,\mu) \leq \mathsf{Sec}_{\mathcal{SE}}^{\varphi_{DTLS}\text{-}\mathsf{FUL}}(k,t+\tilde{c}q,q,\mu) + \mathsf{Sec}_{\mathcal{SE}}^{\$\text{-}\mathsf{CPA}}(k,t+\tilde{c}q,q,\mu).$$

```
A'
proc $\mathsf{Init}_{\varphi_{DTLS}}()$
    $k \leftarrow \mathsf{Init}_{\varphi_{sl}}()$
    $K_\mathsf{F} \leftarrow\!\$\ \mathsf{KGen}_\mathsf{F}(k)$
    $\kappa \leftarrow\!\$\ \{0,1\}^N$
    $seq \leftarrow 0$
    $latest \leftarrow 0$
    $S \leftarrow \varnothing$
    return $k$

proc $\mathsf{Enc}_{\varphi_{DTLS}}(m, ws)$
    if $ws \notin \{8,16\}$ or $l(|m|) < 128$
        return $\perp$
    else
        $seq \leftarrow seq + 1$
        $n \leftarrow \kappa \oplus seq$
        $seq_{partial} \leftarrow seq[N - ws + 1, N]$
        $c \leftarrow \mathsf{Enc}_{\varphi_{sl}}(n, m, seq_{partial})$
        $mask \leftarrow \mathsf{F}(K_\mathsf{F}, c[0,127])$
        $\tilde{n} \leftarrow seq_{partial} \oplus mask[0, ws-1]$
        return $(\tilde{n}, c)$

proc $\mathsf{Dec}_{\varphi_{DTLS}}(\tilde{n}, c)$
    $mask \leftarrow \mathsf{F}(K_\mathsf{F}, c[0,127])$
    $seq_{partial} \leftarrow \tilde{n} \oplus mask[0, |\tilde{n}| - 1]$
    $seq \leftarrow closest(latest, seq_{partial})$
    $n \leftarrow \kappa \oplus seq$
    $m \leftarrow \mathsf{Dec}_{\varphi_{sl}}(n, c, seq_{partial})$
    if $latest - seq > w_r$ or $seq \in S$
        $m \leftarrow \perp$
    if $m \neq \perp$
        $S \leftarrow S \cup seq$
        if $seq > latest$
            $latest \leftarrow seq$
    return $m$

proc $\mathsf{Fin}_{\varphi_{DTLS}}(d)$
    $\mathsf{Fin}_{\varphi_{sl}}(d)$
```

Figure 9.2: The adversary used in the proof of Theorem 9.1.

We prove the following two statements:

$$\mathsf{Sec}_{\mathcal{SE}}^{\varphi_{DTLS}\text{-}\mathsf{FUL}}(k,t,q,\mu) \leq \mathsf{Sec}_{\mathsf{AEAD}}^{\varphi_{sl}\text{-}\mathsf{FUL}}(k, t + cq, q, \mu)$$

$$\mathsf{Sec}_{\mathcal{SE}}^{\$\text{-}\mathsf{CPA}}(k,t,q,\mu) \leq \mathsf{Sec}_{\mathcal{SE}}^{\$\text{-}\mathsf{CPA}}(k,t,q+cq,\mu) + \mathsf{Sec}_{F}^{\mathsf{PRF}\text{-}\mathsf{SEC}}(k,t,q,\mu) + \frac{q(q-1)}{2^{128}}.$$

For the first statement let $\mathcal{A}$ be an adversary for the φ_{DTLS}-FUL game and consider the adversary $\mathcal{A}'$ defined in Figure 9.2. This adversary copies the construction of $\mathcal{SE}$ but replaces the encryptions and decryption with calls to the φ_{sl}-FUL game. As the return values of encryption and decryption queries in this game are exactly the output of the encryption and decryption algorithms, this adversary simulates the φ_{DTLS}-FUL game. So it remains to argue that for any query which would sets *win* to 1 in the simulated φ_{DTLS}-FUL game, *win* is also set in the actual φ_{sl}-FUL game.

Consider a forgery attempt, so a decryption query $(\tilde{n}, c)$ so that $(enc, (\tilde{n}, c)) \notin T$. If c was never the output of the underlying scheme, then this is clearly a successful forgery in the φ_{sl}-FUL game. Similarly, if the nonce n restored from $\tilde{n}$ is not equal to the nonce originally used during encryption, then this also results in a successful forgery for the φ_{sl}-FUL game. Finally, since *mask* is uniquely determined by the ciphertext and $\tilde{n}$ must be the lowest bits of n, there can be no $\tilde{n}' \neq \tilde{n}$ which also restores to n.

So we can assume that no forgery attempts were successful. Now consider a decryption query for a ciphertext $(\tilde{n}, c)$ which was previously encrypted, but is outside of its window but does not decrypt to $\perp$. Note that the nonce n constructed will be different from the one used in encryption, so such a query results in a forgery in the φ_{sl}-FUL game. Note that we require ciphertext uniqueness here, as otherwise the same ciphertext could have been encrypted with a different nonce that has the same lowest digits.

Next consider a decryption query $(\tilde{n}, c)$ for a ciphertext which is sent inside its acceptance window and assume that is has been accepted before, so that $(dec, (\tilde{n}, c), 1) \in T$. Because it is in the acceptance window, its sequence number will be restored correctly, so the replay check will cause the decryption to output $\perp$.

Finally, consider the remaining queries, so queries which are inside their window and not replays. For these the nonce will be restored correctly, so the input to the φ_{sl}-FUL game is the same as the original input for the encryption. An incorrect decryption in the φ_{DTLS}-FUL game therefore can only happen if the decryption in the φ_{sl}-FUL game is incorrect.

The second statement is mostly a direct application of Theorem 6.1 from [BNT19], so we omit most of the details. We start from the \$-CPA$_{\mathcal{SE}}(k, 1)$ game and replace F with an actual random function. This gives the term $\mathsf{Sec}_F^{\mathsf{PRF\text{-}SEC}}(k, t, q, \mu)$. Then we replace the ciphertext c output by AEAD with a random string. Finally, we replace the random function output by random bits. These last two games are only distinguishable if the leading 128 bits of the c are equal, as if this occurs the masks will be equal and the masked sequence numbers can be XORed to get the XOR of the unmasked sequence numbers. This occurs with probability $\frac{q(q-1)}{2^{128}}$.

For the second step we need to add an extra reduction. The construction of [BNT19] does not reduce to the underlying AEAD scheme, but instead to another scheme $\mathcal{SE}'$ where nonces are not assumed to be part of the ciphertext. So we still need to bound the privacy of $\mathcal{SE}'$ by the privacy of AEAD. As the ciphertexts of $\mathcal{SE}'$ only consist of the AEAD output, we can easily construct an adversary which simulates the encryption of $\mathcal{SE}$, forwards its queries to the AEAD game, but only returns the AEAD output. This simulates the game against $\mathcal{SE}'$ with the same value of b, concluding the proof. □

We finish this chapter with some remarks on how the tools from the previous chapters helped in analysing DTLS. First of all, note that simply by defining the predicate φ_{DTLS}, we immediately have a definition for the security notion we should try to achieve, namely \$-$\varphi_{DTLS}$$\mathsf{AE}$. By Theorems 7.9 and 7.10, this notion implies the correctness, integrity and privacy up to correctness.

In the proof that our abstraction of DTLS indeed satisfies \$-$\varphi_{DTLS}$$\mathsf{AE}$, the use of Theorem 7.9 allowed us to prove the tight bound of correctness and integrity in a single step and to separate out the privacy reduction which incurs the additional collision term.

By Theorem 8.3 we also immediately see that DTLS ensures progress-hiding up to an appropriate suppression function. This suppression function has to be fairly strict, but as we argued at the end of Chapter 8, this is not a failing of DTLS, but an inherent property of acceptance windows.

Chapter 10

Conclusion

We began this thesis analysing stateless and stateful encryption schemes. We introduced a variety of notions, including a new notion $-sfAE, which captures privacy using $-type encryption oracles and stateful ciphertext integrity. To motivate this new notion, we also established progress-hiding, another new notion which ensures that an adversary which loses access to a channel for some time cannot tell whether messages were sent during this time. The notions often used for channels, which use LoR encryption oracles, do not ensure progress-hiding, $-sfAE however does. We then showed that the record layer of TLS 1.3 fulfils $-sfAE.

We then introduced predicates, which capture the intended behaviour of a channel in regards to correctness and integrity. This allowed us to extend these ideas to schemes running on unreliable network protocols. For a predicate φ we defined a new notion $\$$-φAE which ensures correctness, integrity and privacy using $-type oracles for any channel characterized by φ. We also established tools which can be used to prove that schemes fulfil this notion and used these tools to show that the record layer of DTLS 1.3 achieves it for an appropriate predicate.

A key take-away is therefore that indistinguishability of ciphertexts from random bits instead of left-or-right indistinguishability is not only desirable in theory, but also yields stronger practical guarantees and is achievable for real-world protocols.

We would like conclude by highlighting some existing research which we believe could be extended by using $-type encryption oracles, giving slightly stronger results, and predicates, allowing easy analysis of schemes like DTLS.

Many channels offer ways to renegotiate keys during communication. Symmetric stateful encryption schemes which offer key updates are formalized in [GM17] using LoR oracles.

The assumption that the error output of encryption schemes is always $\perp$ does not hold in practice. Encryption schemes with multiple error messages are formalized in [BDPS13] using $-type oracles, but only for stateless and stateful schemes.

Another assumption which can lead to real-world attacks is that ciphertexts are delivered atomically instead of as a stream. This has been studied variously for stateful schemes using LoR oracles, for example in [FGMP15], [BDPS12] and [ADHP16].

Finally, for some use-cases it may be desirable to hide the plaintext length by allowing the encryption algorithm encrypt plaintext to ciphertexts of a variable length. This is formalized in [PRS11] for stateful schemes using LoR oracles.

Bibliography

[ADHP16] Martin R. Albrecht, Jean Paul Degabriele, Torben Brandt Hansen, and Kenneth G. Paterson. A surfeit of SSH cipher suites. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1480–1491. ACM Press, October 2016.

[BDJR97] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science*, pages 394–403. IEEE Computer Society Press, October 1997.

[BDJR00] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. https://web.cs.ucdavis.edu/~rogaway/papers/sym-enc.pdf, 2000.

[BDPS12] Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G. Paterson, and Martijn Stam. Security of symmetric encryption in the presence of ciphertext fragmentation. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 682–699. Springer, Heidelberg, April 2012.

[BDPS13] Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G. Paterson, and Martijn Stam. On symmetric encryption with distinguishable decryption failures. Cryptology ePrint Archive, Report 2013/433, 2013. http://eprint.iacr.org/2013/433.

[BDPS14] Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G. Paterson, and Martijn Stam. On symmetric encryption with distinguishable decryption failures. In Shiho Moriai, editor, *Fast Soft-*

ware Encryption – FSE 2013, volume 8424 of *Lecture Notes in Computer Science*, pages 367–390. Springer, Heidelberg, March 2014.

[BHMS15] Colin Boyd, Britta Hale, Stig Frode Mjølsnes, and Douglas Stebila. From stateless to stateful: Generic authentication and authenticated encryption constructions with application to TLS. Cryptology ePrint Archive, Report 2015/1150, 2015. http://eprint.iacr.org/2015/1150.

[BKN02] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm. Cryptology ePrint Archive, Report 2002/078, 2002. http://eprint.iacr.org/2002/078.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, Heidelberg, December 2000.

[BNT19] Mihir Bellare, Ruth Ng, and Björn Tackmann. Nonces are noticed: AEAD revisited. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 235–265. Springer, Heidelberg, August 2019.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, Heidelberg, May / June 2006.

[BSWW13] Christina Brzuska, Nigel P. Smart, Bogdan Warinschi, and Gaven J. Watson. An analysis of the EMV channel establishment protocol. Cryptology ePrint Archive, Report 2013/031, 2013. http://eprint.iacr.org/2013/031.

[DFP+20] Antoine Delignat-Lavaud, Cédric Fournet, Bryan Parno, Jonathan Protzenko, Tahina Ramananandro, Jay Bosamiya, Joseph Lallemand, Itsaka Rakotonirina, and Yi Zhou. A security model and fully verified implementation for the IETF QUIC record layer. Cryptology ePrint Archive, Report 2020/114, 2020. https://eprint.iacr.org/2020/114.

[DR08] Tim Dierks and Eric Rescoria. *RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2*. Internet Engineering Task Force (IETF), August 2008. https://tools.ietf.org/html/rfc5246.

[FGJ20] Marc Fischlin, Felix Günther, and Christian Janson. Robust channels: Handling unreliable networks in the record layers of QUIC and DTLS 1.3. Cryptology ePrint Archive, Report 2020/718, 2020. https://eprint.iacr.org/2020/718.

[FGMP15] Marc Fischlin, Felix Günther, Giorgia Azzurra Marson, and Kenneth G. Paterson. Data is a stream: Security of stream-based channels. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 545–564. Springer, Heidelberg, August 2015.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[GM05] Zvi Gutterman and Dahlia Malkhi. Hold your sessions: An attack on Java session-id generation. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 44–57. Springer, Heidelberg, February 2005.

[GM17] Felix Günther and Sogol Mazaheri. A formal treatment of multi-key channels. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 587–618. Springer, Heidelberg, August 2017.

[KPB03] Tadayoshi Kohno, Adriana Palacio, and John Black. Building secure cryptographic transforms, or how to encrypt and MAC. Cryptology ePrint Archive, Report 2003/177, 2003. http://eprint.iacr.org/2003/177.

[PRS11] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 372–389. Springer, Heidelberg, December 2011.

[RBBK01] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001: 8th Conference on Computer and Communications Security*, pages 196–205. ACM Press, November 2001.

[Res18] Eric Rescoria. *RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3*. Internet Engineering Task Force (IETF), August 2018. https://tools.ietf.org/html/rfc8446.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM CCS 2002: 9th Conference on Computer and Communications Security*, pages 98–107. ACM Press, November 2002.

[Rog04] Phillip Rogaway. Nonce-based symmetric encryption. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption – FSE 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 348–359. Springer, Heidelberg, February 2004.

[RS06] Phillip Rogaway and Thomas Shrimpton. Deterministic authenticated-encryption: A provable-security treatment of the key-wrap problem. Cryptology ePrint Archive, Report 2006/221, 2006. http://eprint.iacr.org/2006/221.

[RZ18] Phillip Rogaway and Yusi Zhang. Simplifying game-based definitions: Indistinguishability up to correctness and its application to stateful AE. Cryptology ePrint Archive, Report 2018/558, 2018. https://eprint.iacr.org/2018/558.

[Shr04] Tom Shrimpton. A characterization of authenticated-encryption as a form of chosen-ciphertext security. Cryptology ePrint Archive, Report 2004/272, 2004. http://eprint.iacr.org/2004/272.