Overloading the Nonce: Rugged PRPs, Nonce-Set AEAD, and Order-Resilient Channels*

Jean Paul Degabriele[1,2] and Vukašin Karadžić[2]

[1] Technology Innovation Institute, UAE
jeanpaul.degabriele@tii.ae
[2] Technische Universität Darmstadt, Germany
vukasin.karadzic@tu-darmstadt.de

Abstract. We introduce a new security notion that lies right in between pseudorandom permutations (PRPs) and strong pseudorandom permutations (SPRPs). We call this new security notion and any (tweakable) cipher that satisfies it a *rugged pseudorandom permutation* (RPRP). Rugged pseudorandom permutations lend themselves to some interesting applications, have practical benefits, and lead to novel cryptographic constructions. Our focus is on variable-length tweakable RPRPs, and analogous to the encode-then-encipher paradigm of Bellare and Rogaway, we can generically transform any such cipher into different AEAD schemes with varying security properties. However, the benefit of RPRPs is that they can be constructed more efficiently as they are weaker primitives than SPRPs (the notion traditionally required by the encode-then-encipher paradigm). We can construct RPRPs using only two layers of processing, whereas SPRPs typically require three layers of processing over the input data. We also identify a new transformation that yields RUP-secure AEAD schemes with more compact ciphertexts than previously known. Further extending this approach, we arrive at a new generalized notion of authenticated encryption and a matching construction, which we refer to as *nonce-set AEAD*. Nonce-set AEAD is particularly well-suited in the context of secure channels, like QUIC and DTLS, that operate over unreliable transports and employ a window mechanism at the receiver's end of the channel. We conclude by presenting a generic construction for transforming a nonce-set AEAD scheme into an order-resilient secure channel. Our channel construction sheds new light on order-resilient channels and additionally leads to more compact ciphertexts when instantiated from RPRPs.

Keywords: Tweakable Ciphers · Rugged Pseudorandom Permutations · AEAD · Secure Channels

Table of Contents

1 Introduction

The modern view of symmetric encryption follows a nonce-based syntax. At first, this may seem like a superficial detail but it has important ramifications both practically and theoretically. When first conceived by Rogaway in [29], its primary motivation was to position the security of symmetric encryption on more solid ground by lifting its reliance on good sources of randomness. It thus replaced an initialization vector, required to be uniformly random, for a nonce that instead is only required to never repeat. Besides significantly reducing susceptibility to implementation errors, it added versatility by elegantly aligning the two main flavours of symmetric encryption— randomized and stateful—into a single unified syntax from which they can easily be realized. The resistance to misuse was later fortified in the strengthened security notion by Rogaway and Shrimpton in [30]. On the more theoretical side, this seemingly minor syntactical change has major consequences on how symmetric encryption and message authentication compose together to form authenticated encryption. In contrast to the traditional view that only encrypt-then-MAC results in a generically secure composition [7], all three composition paradigms become secure under the nonce-based syntax and the mild requirement of tidiness [25].

Secure Channels. A major application of nonce-based AEAD is to realize secure channels in protocols like TLS, SSH, and QUIC. Here, a number of options arise on how to handle the nonce, initialize it, update it, and communicate it to the other party. Typically, secure channels need to protect against the replay and reordering of ciphertexts, which in turn necessitates the receiver to be stateful [6]. Accordingly, a common approach is to initialize the nonce to a common value and each party increments it (independently) upon every encryption and decryption. This works well as long as the transport protocol, upon which the secure channel is realized, is reliable and order-preserving, meaning that ciphertexts are delivered in the same order as they were sent and without being lost. TLS and SSH operate over TCP, which is reliable and order-preserving, but at the same time introduces issues such as head-of-line blocking[1] which degrades performance. This motivated the emergence of protocols like DTLS and QUIC, which operate over UDP, thereby avoiding head-of-line blocking at the expense of having to deal with out-of-order delivery and dropped ciphertexts.

Operating secure channels over UDP means that the receiver cannot predict the nonce as ciphertexts may arrive out of order. Accordingly, the nonce has to be communicated together with each ciphertext. Moreover, if the nonce is set to be a message number, the receiver can use it to recover the correct ordering of the messages. In fact, because in nonce-based AEAD the nonce is implicitly authenticated, the above approach works even against adversarial reordering strategies. Indeed, this is roughly the approach adopted in DTLS 1.3 and QUIC. Thus, while the nonce was originally only intended to diversify ciphertexts, in these protocols it is 'overloaded' to additionally serve a secondary purpose for recovering the correct message ordering. This is yet another example of the beauty and versatility of a well-crafted definition like nonce-based AEAD. However, attaching the nonce to the ciphertext in the clear exposes metadata which can undermine privacy [13] and possibly confidentiality [8]. Accordingly QUIC and DTLS 1.3 separately encrypt the nonce before attaching it to the ciphertext. In turn this has led to the notion of nonce-hiding AEAD [8], an idea that can be traced back to Bernstein [10].

Encode-then-Encipher. A classical technique for constructing an authenticated encryption scheme is the encode-then-encipher paradigm by Bellare and Rogaway [9]. The technique builds an authenticated encryption scheme from a variable-input-length cipher by properly encoding the message with randomness and redundancy in order to obtain confidentiality and integrity. A more modern take on the encode-then-encipher paradigm was put forth by Shrimpton and Terashima in [32] where it was extended to obtain nonce-based authenticated encryption with associated data (AEAD) from tweakable variable-input-length ciphers. A noteworthy feature of the encode-then-encipher paradigm is that it yields AEAD schemes that satisfy the strongest possible security—misuse resistance [30] and release-of-unverified plaintext (RUP) security [1, 3, 21] simultaneously. Despite their strong security, such schemes are scarce in real-world systems. In all likelihood, this is due to tweakable ciphers generally being heavy primitives whose performance lags behind that of more efficient AEAD schemes. In this respect, one exception is AEZ [21] which offers competitive speeds although requiring three layers of processing. However its security relies on a non-standard heuristic analysis and, in addition, it is also a significantly complex scheme to implement.

[1] https://en.wikipedia.org/wiki/Head-of-line_blocking

1.1 Contribution

Rugged Pseudorandom Permutations. Our first contribution is a novel security definition for tweakable ciphers that sits between a pseudorandom permutation and a strong pseudorandom permutation. The security definition assumes a cipher defined over a 'split' domain, meaning that its inputs and outputs will typically consist of a pair of strings, possibly of different sizes, rather than a single string. A salient characteristic of our security definition is that it imposes stronger security requirements on the enciphering algorithm than on the deciphering algorithm. Intuitively, we will still require an adversary to distinguish between the cipher and a random permutation. However, while the adversary will have full access to the enciphering algorithm its access to the deciphering algorithm will be restricted, thereby giving rise to the asymmetric security between the two algorithms. Due to the uneven domain and the asymmetry in the cipher's security we choose to call such a cipher a rugged pseudorandom permutation (RPRP).

The benefit of this security definition is that it strikes a new balance in which security is sufficiently weakened to allow for more efficient cipher constructions while still being strong enough to be of use in practice. Our RPRP construction is inspired by the PIV construction by Terashima and Shrimpton [32] and the GCM-RUP construction by Ashur, Dunkelman, and Luykx [2]. Our construction, Unilaterally-Protected IV (`UIV`), is directly obtained from the PIV construction by shaving off its last layer. GCM-RUP is similarly derived from PIV by shaving off the first layer and then augmenting it to obtain a nonce-based AEAD scheme that is RUP secure. Like GCM-RUP, `UIV` can be instantiated from GCM components and benefit from GCM's now-ubiquitous hardware support that enables its superior performance. The benefit of drawing the boundary around `UIV` is that firstly it is a length-preserving cipher which is advantageous in settings such as disk encryption. Secondly, it is a more versatile primitive which, as we shall see, can be easily augmented to yield different AEAD schemes. Indeed, one specific transformation recovers GCM-RUP, but our general treatment allows us to uncover several new AEAD schemes with differing properties and improvements.

Constructing AEAD From RPRPs. We revisit the encode-then-encipher paradigm in the context of RPRPs. The asymmetry in the RPRP security definition prompts us to consider two variations of this paradigm: Encode-then-Encipher (`EtE`) and Encode-then-Decipher (`EtD`), where the latter uses the deciphering algorithm to encrypt and the enciphering algorithm to decrypt. We show that `EtE` yields misuse-resistant AEAD and that `EtD` yields RUP-secure AEAD. A notable instantiation of the encode-then-encipher paradigm is to 'overload' the use of the nonce to additionally serve as the redundancy in the encoding that provides integrity. This approach appears to have been missed in prior works. For instance, GCM-RUP simultaneously encrypts the nonce and adds redundancy in the message, resulting in an unnecessary expansion in the ciphertext. On the other hand, when `EtD` is instantiated this way with `UIV` we obtain a RUP-secure scheme with more compact ciphertexts than GCM-RUP.

Nonce-Set AEAD and its Construction From RPRPs. Taking this idea of overloading the nonce for integrity a step further, we arrive at a new AEAD construction with novel functionality. This functionality is motivated by the use case of AEAD in secure channels like QUIC and DTLS. We formalize this functionality as a new primitive that we call nonce-set AEAD, which extends and generalizes the standard definition of nonce-based AEAD. Nonce-set AEAD alters the decryption algorithm to additionally take a set of nonces instead of a single one. Intuitively decryption will succeed if the correct nonce is among this set. Moreover, the decryption algorithm will return the nonce in the supplied set that was deemed correct as part of its output. We show how to generically construct such a scheme from an RPRP through a construction we call Authenticate-with-Nonce (`AwN`) and show that it even achieves misuse-resistance AEAD security. The `AwN` construction requires a mechanism for representing nonce-sets compactly and efficiently testing for membership in this set. Of course, since any SPRP is automatically an RPRP, `AwN` can also be instantiated using other well-known SPRP constructions.

Order-Resilient Secure Channels From Nonce-Set AEAD. In order-resilient channels, the nonce is often overloaded to serve as a message number that can be used to recover the correct ordering of the decrypted messages. Nonce-set AEAD facilitates such an approach and can be plugged in directly with the window mechanisms that are used in real-world protocols like QUIC and DTLS. Such window mechanisms can be fairly complex and hard to understand when presented as code. Moreover, they affect the security of the channel, and as a result, analyzing the security of these channels can become rather

daunting at times. Our treatment based on nonce-set AEAD will help tame this complexity. The other reason for introducing nonce-set AEAD is that it will allow for more bandwidth-efficient constructions from RPRPs by additionally overloading the nonce to provide integrity in a way that is compatible with the window mechanisms in the channel.

Recent work by Fischlin, Günther, and Janson [17] introduces a formal framework for analyzing the security of order-resilient secure channels like QUIC and DTLS. Central to the framework is a support predicate that expresses the expected behaviour of such channels. Many possibilities exist here in terms of how much reordering should be tolerated, the specific window mechanism to use, and how to handle replays, but the support predicate neatly captures these variations in their full generality. We build on the framework in [17] to show how to generically transform any nonce-set AEAD scheme into a secure channel for any support predicate that may be required. Besides having practical value, that of offering order-resilient secure channels with more compact ciphertexts, our construction is also instructive in that it decomposes the structure of complex secure channels into a handful of much simpler and manageable components. It should be noted that nonce-set AEAD can also be realized through other constructions—such as the nonce-hiding schemes in [8]. As such, our approach is very general and versatile.

1.2 Relation to Counter Galois Onion

This work stemmed out from other work, concurrent to this one, on the design of Counter Galois Onion (CGO), a proposal for a new onion encryption scheme for Tor [15]. Under the hood, CGO employs an extended Rugged PRP to process each layer of encryption. In particular, the notion of a Rugged PRP was developed in both works in parallel and went through a number of iterations. It was initially conceived as an abstraction to facilitate the security proof of CGO, but we later realised that it had applications beyond onion encryption which motivated the research in this paper.

2 Preliminaries

Notation. For any non-negative integer $n \in \mathbb{N}$, $\{0,1\}^n$ denotes the set of bit strings of size n, $\{0,1\}^*$ denotes the set of all finite binary strings, and $\{0,1\}^{\geq n}$ denotes the set of all finite bit strings of size greater or equal to n. The empty string is denoted by ε. For any string X, $|X|$ denotes its length in bits. Then for any non-negative integer $n \leq |X|$, $\lfloor X \rfloor_n$ and $\lceil X \rceil_n$ denote respectively the substrings of the leftmost and rightmost n bits of X, and $X \ll n$ denotes the bit string of size $|X|$ obtained by truncating its leftmost n bits and appending n zeros to its right. For any two strings X and Y, of lengths $|X| = n$ and $|Y| = m$, where $n < m$, $X \oplus Y$ denotes the operation of appending $m - n$ zeros to the left of X, and then XORing the expanded string X with Y. For any pair of strings (X, Y) we define their combined length $|(X, Y)|$ as $|X| + |Y|$ and we use $\langle X, Y \rangle$ to denote an injective mapping from string pairs into single strings.

For any set $\mathcal{S}$, we use $|\mathcal{S}|$ to denote its cardinality, $\mathcal{P}(\mathcal{S})$ to denote its power set, i.e., the set of all its subsets, and $\mathrm{Perm}[\mathcal{S}]$ to denote the set of all permutations over the elements of $\mathcal{S}$. The empty set is denoted by $\emptyset$. For any two sets $\mathcal{T}$ and $\mathcal{X}$, $\mathsf{IC}(\mathcal{T}, \mathcal{X})$ denotes the set of all ciphers over the domain $\mathcal{X}$ and key space $\mathcal{T}$, $\mathrm{Func}(\mathcal{X}, \infty)$ denotes the set of all functions mapping elements in $\mathcal{X}$ to elements in $\{0,1\}^\infty$, and $\pm\mathrm{Func}(\mathcal{T}, \mathcal{X})$ denotes the set of all functions mapping elements in $\{+,-\} \times \mathcal{T} \times \mathcal{X}$ to elements in $\mathcal{X}$. In our pseudocode we use lists as an abstract data type. We use $[\,]$ to denote the empty list, and for any two lists $\mathcal{L}_1$ and $\mathcal{L}_2$, we use $\mathcal{L}_1 \| \mathcal{L}_2$ to denote the list obtained by appending $\mathcal{L}_2$ to $\mathcal{L}_1$. Lists are indexed starting at position zero, and $\mathcal{L}_1[i]$ denotes the element in $\mathcal{L}_1$ at position i. For a string X and a list $\mathcal{L}$, the function $\mathrm{index}(X, \mathcal{L})$ returns the smallest index in $\mathcal{L}$ in which X is located, if X is contained in $\mathcal{L}$, and returns $\perp$ otherwise.

For events E and F, we use $\neg E$ to denote the complement event of E, $\Pr[E]$ to denote the probability of E, and $\Pr[E \mid F]$ to denote the probability of E conditioned on F. Finally, $\Pr[P : E]$ denotes the probability of E occuring after executing some random process P.

Tweakable Ciphers. A tweakable cipher is an algorithm

$$\widetilde{\mathsf{EE}} : \mathcal{K} \times \mathcal{T} \times \mathcal{X} \to \mathcal{X}$$

such that for any $(K, T) \in \mathcal{K} \times \mathcal{T}$ the mapping $\widetilde{\mathsf{EE}}(K, T, \cdot)$ identifies a permutation over the elements in $\mathcal{X}$. We refer to $\mathcal{K}$ as the key space, $\mathcal{T}$ as the tweak space, and $\mathcal{X}$ as the domain. We use $\widetilde{\mathsf{EE}}_K(T, \cdot)$ as

$\mathsf{Ver}_K(N,H,C)$	$\$(N,H,M)$	$\bot(N,H,C)$
$M \leftarrow \mathsf{Dec}_K(N,H,C)$	$C \leftarrow\!\$\, \{0,1\}^{\mathsf{clen}(\lvert N\rvert,\lvert H\rvert,\lvert M\rvert)}$	**return** $\bot$
if $M \in \mathcal{M}$	**return** C	
$\quad M \leftarrow \top$		
return M		

Fig. 1: Oracles used to define nAE, MRAE, and RUPAE security.

shorthand for $\widetilde{\mathsf{EE}}(K,T,\cdot)$ and $\widetilde{\mathsf{EE}}_K^{-1}(T,\cdot)$ to denote the corresponding inverse permutation. A tweakable cipher is required to be length preserving, meaning that for all $(K,T,X) \in \mathcal{K} \times \mathcal{T} \times \mathcal{X}$ it holds that $|\widetilde{\mathsf{EE}}_K(T,X)| = |X|$. We also refer to $\widetilde{\mathsf{EE}}$ and $\widetilde{\mathsf{EE}}^{-1}$ as the enciphering and deciphering algorithms of the tweakable cipher. In the special case where $\mathcal{X} = \{0,1\}^n$, for some positive integer n, we call the cipher a tweakable blockcipher and denote it by $\widetilde{\mathsf{E}}$. Thus we generally reserve $\widetilde{\mathsf{EE}}$ to denote a variable-input-length tweakable cipher, which may itself be constructed from an underlying tweakable blockcipher $\widetilde{\mathsf{E}}$.

Security. The security definition for tweakable ciphers is provided in Appendix A.1.

Nonce-Based AEAD. A nonce-based encryption scheme $\mathtt{SE} = (\mathsf{Enc}, \mathsf{Dec})$ is a pair of algorithms to which we associate a key space $\mathcal{K}$, a nonce space $\mathcal{N}$, a header (associated data) space $\mathcal{H}$, a message space $\mathcal{M}$ and a ciphertext space $\mathcal{C}$, all of which are subsets of $\{0,1\}^*$. The encryption algorithm Enc and the decryption algorithm Dec are both deterministic and their syntax is given by

$$\mathsf{Enc} : \mathcal{K} \times \mathcal{N} \times \mathcal{H} \times \mathcal{M} \to \mathcal{C} \text{ and } \mathsf{Dec} : \mathcal{K} \times \mathcal{N} \times \mathcal{H} \times \mathcal{C} \to \mathcal{M} \cup \{\bot\}\,.$$

The special symbol $\bot$ serves to indicate that the decryption algorithm deemed its input to be invalid. A nonce-based encryption scheme is required to be *correct* and *tidy* [25]. Correctness requires that for all $(K,N,H,M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{H} \times \mathcal{M}$ it must hold that

$$\mathsf{Dec}_K(N,H,\mathsf{Enc}_K(N,H,M)) = M.$$

Tidiness, on the other hand, requires that for any $(K,N,H,C) \in \mathcal{K} \times \mathcal{N} \times \mathcal{H} \times \mathcal{C}$

$$\text{if } \mathsf{Dec}_K(N,H,C) \neq \bot \text{ then } \mathsf{Enc}_K(N,H,\mathsf{Dec}_K(N,H,C)) = C.$$

We further require that encryption be length-regular, meaning that the size of ciphertexts depend only on the *sizes* of N, H and M. Accordingly, we associate to every nonce-based AEAD scheme a ciphertext length function clen, mapping the triple $(|N|,|H|,|M|)$ to the ciphertext length in bits.

Security. A nonce-based encryption scheme is said to be AEAD if it additionally satisfies (nAE) security. We use a variant of nAE from [5] which is equivalent to the usual formulation. Namely we require that no efficient adversary be able to distinguish between oracle access to the real encryption algorithm $\mathsf{Enc}_K(\cdot,\cdot,\cdot)$ and the real *verification algorithm* $\mathsf{Ver}_K(\cdot,\cdot,\cdot)$ (defined in Fig. 1) from their corresponding idealisations $\$(\cdot,\cdot,\cdot)$ and $\bot(\cdot,\cdot,\cdot)$. Throughout this distinguishing game, the adversary is required to be *nonce-respecting*, meaning that it never repeats nonce values across encryption queries, and must not *forward* queries from the encryption oracle to the decryption oracle, meaning that it cannot make a query (N,H,C) if it previously queried (N,H,M) and got C in return.

Definition 1 (nAE Advantage). *Let* $\mathtt{SE} = (\mathsf{Enc}, \mathsf{Dec})$ *be a nonce-based encryption scheme and let* $\mathcal{A}$ *be a nonce-respecting adversary that does not make forwarding queries. Then the* nAE *advantage of* $\mathcal{A}$ *with respect to* $\mathtt{SE}$ *is defined as*

$$\mathbf{Adv}^{\mathrm{nae}}_{\mathtt{SE}}(\mathcal{A}) = \left|\Pr\left[K \leftarrow\!\$\, \mathcal{K} : \mathcal{A}^{\mathsf{Enc}_K(\cdot,\cdot,\cdot),\mathsf{Ver}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot,\cdot),\bot(\cdot,\cdot,\cdot)} \Rightarrow 1\right]\right|.$$

The stronger notion of misuse-resistant AEAD MRAE is defined analogously by replacing the requirement on the adversary that it be nonce-respecting with the requirement that it never repeat an encryption query.

Definition 2 (MRAE **Advantage**). *Let* $\mathtt{SE} = (\mathsf{Enc}, \mathsf{Dec})$ *be a nonce-based encryption scheme and let* $\mathcal{A}$ *be an adversary that never repeats encryption queries and does not make forwarding queries. Then the* MRAE *advantage of* $\mathcal{A}$ *with respect to* SE *is defined as*

$$\mathbf{Adv}^{\mathrm{mrae}}_{\mathtt{SE}}(\mathcal{A}) = \left|\Pr\left[K \leftarrow\!\$\, \mathcal{K} : \mathcal{A}^{\mathsf{Enc}_K(\cdot,\cdot,\cdot),\mathsf{Ver}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right]\right|.$$

Release of Unverified Plaintext. In practice, in the event of a decryption failure, the decryption algorithm may leak more information than what is captured by the standard security notions. Prior works proposed strengthened notions which modelled such leakage as distinguishable decryption failures [11], release of unverified plaintexts (RUP) [1], and robust authenticated encryption [21]. Then in [3] Barwell et al. introduced *subtle* authenticated encryption to compare and unify these three security models. Here we will utilise the RUPAE security definition as defined by Barwell et al. through their subtle AE framework.

Subtle AE. (*c.f.* [3]) A *subtle encryption scheme* $\mathtt{SSE} = (\mathsf{Enc}, \mathsf{Dec}, \Lambda)$ is a nonce-based encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ augmented with a (deterministic) decryption leakage function Λ intended to model the protocol leakage from decryption failures. The leakage function takes the same inputs as the decryption algorithm but instead returns either a leakage string or the special symbol $\top$. The symbol $\top$ indicates that decryption was successful, and thus for any subtle encryption scheme it must hold that for any K, N, H and C exactly one of the following be true:

$$\perp \leftarrow \mathsf{Dec}_K(N, H, C) \quad \text{or} \quad \top \leftarrow \Lambda_K(N, H, C).$$

That is, for any input either decryption returns $\perp$ and a leakage string is returned by Λ, or decryption succeeds thereby returning the full plaintext but Λ returns no leakage string. In practice the leakage depends on how the scheme is implemented, how it is integrated into the larger system, and the scheme itself. Thus a subtle encryption scheme aims to model any potential leakage, via Λ, in order to show that the underlying scheme remains secure even in the presence of this additional leakage. This is formalised through the following security notion.

Security. In rough terms, RUPAE security can be understood as extending nAE security by additionally giving the adversary oracle access to the decryption leakage function. For a subtle encryption scheme to be RUPAE secure we then require the existence of a corresponding leakage simulator S which can simulate this leakage in the ideal world for any adversary. Intuitively, if the leakage function can be simulated without the secret key it is of no use to the adversary.

Definition 3 (RUPAE **Advantage**). *Let* $\mathtt{SSE} = (\mathsf{Enc}, \mathsf{Dec}, \Lambda)$ *be a subtle AE encryption scheme and let* $\mathcal{A}$ *be a nonce-respecting adversary that does not forward encryption queries to the decryption and leakage oracles. Then the advantage of* $\mathcal{A}$ *with respect to* SSE *and the leakage simulator* S *is defined as*

$$\mathbf{Adv}^{\mathrm{rupae}}_{\mathtt{SSE}}(\mathcal{A}, \mathsf{S}) = \left|\Pr\left[K \leftarrow\!\$\, \mathcal{K} : \mathcal{A}^{\mathsf{Enc}_K(\cdot,\cdot,\cdot),\mathsf{Dec}_K(\cdot,\cdot,\cdot),\Lambda_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot,\cdot),\perp(\cdot,\cdot,\cdot),\mathsf{S}(\cdot,\cdot,\cdot)} \Rightarrow 1\right]\right|.$$

Nonce-Hiding AEAD. In Appendix A.2 we cover the syntax of nonce-hiding AEAD and how the security definitions covered so far can be adapted to that setting.

Encodings and Redundancy Functions. In the encode-then-encipher paradigm one typically requires some encoding scheme that maps messages to some sparse set of strings [9, 32]. In our case, we will additionally require the ability to "localize" the redundancy within the encoding. Accordingly we will instead use a redundancy function for generating the redundancy which will then be joined to the message to form the encoded input to the tweakable cipher. More specifically, this redundancy function will satisfy one of the following two syntaxes:

$$\mathsf{Func}_2 : \mathcal{N} \times \mathcal{H} \to \mathcal{X}$$

or

$$\mathsf{Func}_3 : \mathcal{N} \times \mathcal{H} \times \mathcal{M} \to \mathcal{X}.$$

Furthermore, we will require Func_3 to be collision resistant over inputs with distinct nonces. We say that Func_3 is (δ, t)-*collision resistant* if for all efficient adversaries $\mathcal{A}$ running in time t it holds that:

$$\Pr[((N, H, M), (N', H', M')) \leftarrow \mathcal{A} : \mathsf{Func}_3(N, H, M) = \mathsf{Func}_3(N', H', M') \wedge N \neq N'] \leq \delta.$$

3 Rugged Pseudorandom Permutations

We now introduce a new security notion for tweakable ciphers that provides intermediate security. We call this notion, and by extension, any tweakable cipher that satisfies it a *rugged pseudorandom permutation* (RPRP). A distinctive characteristic of RPRPs is that they are tweakable ciphers over a split domain $\mathcal{X}_L \times \mathcal{X}_R$, where we refer to $\mathcal{X}_L$ as the *left set* and $\mathcal{X}_R$ as the *right set*. Note that the split domain is an implicit requirement of the security definition which would not make sense otherwise. We will typically let $\mathcal{X}_L = \{0,1\}^n$ and $\mathcal{X}_R = \{0,1\}^{\geq m}$ for some non-negative integers n and m, but other choices are possible. Furthermore, for ease of notation, we will simply write $\widetilde{\mathsf{EE}}_K(T, X_L, X_R)$ instead of $\widetilde{\mathsf{EE}}_K(T, (X_L, X_R))$ and apply the same rule to $\widetilde{\mathsf{EE}}^{-1}$.

For sufficiently large n, RPRP security sits right in between PRP security and SPRP security. This is achieved by giving the adversary only partial access to the decipher algorithm. This partial access is provided via two separate oracles, a partial *decipher* oracle and a *guess* oracle. Each oracle limits access to the decipher algorithm in a different way. The decipher oracle severely restricts the set of values on which it can be queried. In contrast, the guess oracle imposes no significant restrictions on the inputs, but it only returns a single bit of information. The combined effect of these restrictions is to relax the extent to which the decipher algorithm needs to be pseudorandom. As a result, there is an asymmetry between the encipher and decipher algorithms in that the former is required to be more pseudorandom than the latter. The term *rugged* in the name is meant to reflect this asymmetry in security and the uneven split in the domain.

The full formal security definition is presented in the next subsection. As we will show in later sections, this notion suffices to generically transform any tweakable cipher that satisfies it into an AEAD scheme with strong security properties. In Sections 4 and 5.3 we present three such transformations. At the same time, the notion is significantly weaker than strong pseudorandom permutations as it allows for more efficient constructions. Strong pseudorandom permutations typically require three layers of processing, where each layer consists of processing the data through a block cipher or a universal hash, and both enciphering and deciphering are two-pass algorithms. In contrast, the `UIV` construction which we present in this section consists of two processing layers where enciphering is a two-pass algorithm but deciphering requires only a single pass over the data as the two layers can be processed in parallel. Admittedly some of the definitional choices, particularly the restrictions imposed on the decipher oracle and the introduction of the guess oracle, in the RPRP definition may seem arbitrary at first. Part of the rationale behind these definitional choices is to require the bare minimum from the tweakable cipher to make the generic transformations, shown in Sections 4 and 5.3, go through.

3.1 RPRP Security

Let $\widetilde{\mathsf{EE}}$ be a tweakable cipher over a split domain $\mathcal{X}_L \times \mathcal{X}_R$ with an associated key space $\mathcal{K}$ and tweak space $\mathcal{T}$. Then for any cipher $\widetilde{\mathsf{EE}}$, RPRP security is defined via the RPRP game shown in Fig. 2. Here the adversary is given access to either the real tweakable cipher construction $\widetilde{\mathsf{EE}}$ or an ideal cipher $\widetilde{\Pi}$ and its task is to determine which of the two it is interacting with. It interacts with the cipher through three oracles: *encipher* (En), *decipher* (De), and *guess* (Gu).

The En oracle provides full access to the encipher algorithm, whereas De provides only partial access to the decipher algorithm. In De access is restricted by checking Y_L for membership in the sets $\mathcal{F}$ and $\mathcal{R}$ and then suppressing the output (via $\lightning$) when this is the case. This check translates to two types of decipher queries that the adversary cannot make. The first is a decipher query where the left value was previously output by the encipher oracle. That is, if an encipher query was made such that $(Y_L, Y_R) \leftarrow \textsc{En}(T, X_L, X_R)$, then no query of the form $\textsc{De}(T', Y_L, Y'_R)$ is allowed for any values of T' and Y'_R. The second is a decipher query that repeats a left value from a prior decipher query. Namely, a query $\textsc{De}(T, Y_L, Y_R)$ when a query of the form $\textsc{De}(T', Y_L, Y'_R)$, for some T' and Y'_R, was already made.

The Gu oracle provides an additional interface to the decipher algorithm. It takes an input to the decipher algorithm together with a set of guesses $\boldsymbol{V}$ for the corresponding left output. In the real world, Gu returns a boolean value indicating whether any of the guesses is correct, whereas it always returns **false** in the ideal world. To avoid trivial-win conditions, we need to restrict the adversary to only make guess queries for which it does not already know the answer. Accordingly, guess queries are required to be "unused", meaning that they have not been already queried on De or returned by En. The set $\mathcal{U}$ serves to keep track of used triples (T, Y_L, Y_R) and suppress the output in Gu when such a query is detected. Finally, the game is parametrized by a positive integer v, limiting the size of $\boldsymbol{V}$ in every query. We quantify the RPRP security of a tweakable cipher via the usual advantage measure shown below.

Game $\mathsf{RPRP}^{\mathcal{A},v}_{\widetilde{\mathsf{EE}}}$	$\textsc{De}(T, Y_L, Y_R)$
$K \leftarrow\!\$\, \mathcal{K}$	**if** $Y_L \in \mathcal{F} \cup \mathcal{R}$
$b \leftarrow\!\$\, \{0,1\}$	**return** $\lightning$
$\mathcal{F}, \mathcal{R}, \mathcal{U} \leftarrow \emptyset, \emptyset, \emptyset$	**if** $b = 0$
$\widetilde{\Pi} \leftarrow\!\$\, \mathsf{IC}(\mathcal{T}, \mathcal{X}_L \times \mathcal{X}_R)$	$(X_L, X_R) \leftarrow \widetilde{\Pi}^{-1}(T, Y_L, Y_R)$
$b' \leftarrow \mathcal{A}^{\textsc{En},\textsc{De},\textsc{Gu}}$	**else**
return $b = b'$	$(X_L, X_R) \leftarrow \widetilde{\mathsf{EE}}_K^{-1}(T, Y_L, Y_R)$
	$\mathcal{R} \xleftarrow{\cup} \{Y_L\}$; $\mathcal{U} \xleftarrow{\cup} \{(T, Y_L, Y_R)\}$
	return (X_L, X_R)
$\textsc{En}(T, X_L, X_R)$	$\textsc{Gu}(T, Y_L, Y_R, \boldsymbol{V})$
if $b = 0$	**if** $((T, Y_L, Y_R) \in \mathcal{U}) \vee (\lvert\boldsymbol{V}\rvert > v)$
$(Y_L, Y_R) \leftarrow \widetilde{\Pi}(T, X_L, X_R)$	**return** $\lightning$
else	**if** $b = 0$
$(Y_L, Y_R) \leftarrow \widetilde{\mathsf{EE}}_K(T, X_L, X_R)$	**return false**
$\mathcal{F} \xleftarrow{\cup} \{Y_L\}$; $\mathcal{U} \xleftarrow{\cup} \{(T, Y_L, Y_R)\}$	**else**
return (Y_L, Y_R)	$(X_L, X_R) \leftarrow \widetilde{\mathsf{EE}}_K^{-1}(T, Y_L, Y_R)$
	return $X_L \in \boldsymbol{V}$

Fig. 2: The game used to define RPRP security for a tweakable cipher $\widetilde{\mathsf{EE}}$.

Definition 4 (RPRP Advantage). *Let $\widetilde{\mathsf{EE}}$ be a tweakable cipher over a split domain $(\mathcal{X}_L \times \mathcal{X}_R)$. Then for a positive integer v and an adversary $\mathcal{A}$ attacking the* RPRP *security of $\widetilde{\mathsf{EE}}$ the corresponding advantage is defined as*

$$\mathbf{Adv}^{\mathrm{rprp}}_{\widetilde{\mathsf{EE}}}(\mathcal{A}, v) = \left| 2 \Pr\left[\mathsf{RPRP}^{\mathcal{A},v}_{\widetilde{\mathsf{EE}}} \Rightarrow 1\right] - 1 \right|.$$

3.2 Unilaterally-Protected IV (UIV)

We next present a variable-input-length tweakable cipher construction, called Unilaterally-Protected IV (UIV), that achieves RPRP security. It is easily derived from the three-round Protected IV construction from [32] by simply eliminating the last layer and using a slightly different abstraction. Shrimpton and Terashima noted that all three rounds are necessary for SPRP security, but as we show in Theorem 1, two rounds suffice for RPRP security. The construction is composed of a tweakable blockcipher $\widetilde{\mathsf{E}}$ over the domain $\mathcal{X}_L = \{0,1\}^n$ with tweak space $\mathcal{T} \times \mathcal{X}_R$ and a matching variable-output-length pseudorandom function F with domain $\mathcal{X}_L$ and range $\mathcal{X}_R$. The tweak space of the resulting UIV cipher is $\mathcal{T}$. A pseudocode description of the construction is given in Fig. 3 and Fig. 4 shows a graphical representation of its encipher algorithm. The RPRP security of the UIV construction is stated formally in Theorem 1, the proof of which can be found in Appendix B.2.

Theorem 1. *Let* UIV *be the construction defined in Fig. 3 over the domain $\{0,1\}^n \times \{0,1\}^{\geq m}$. For a positive integer v and an adversary $\mathcal{A}$ making q_{en} encipher queries, q_{de} decipher queries and q_{gu} guess*

$\widetilde{\mathsf{EE}}_{K1,K2}(T, X_L, X_R)$	$\widetilde{\mathsf{EE}}^{-1}_{K1,K2}(T, Y_L, Y_R)$
$Y_L \leftarrow \widetilde{\mathsf{E}}_{K1}((T, X_R), X_L)$	$X_R \leftarrow \mathsf{F}_{K2}(Y_L, \lvert Y_R\rvert) \oplus Y_R$
$Y_R \leftarrow \mathsf{F}_{K2}(Y_L, \lvert X_R\rvert) \oplus X_R$	$X_L \leftarrow \widetilde{\mathsf{E}}^{-1}_{K1}((T, X_R), Y_L)$
return (Y_L, Y_R)	**return** (X_L, X_R)

Fig. 3: Pseudocode description of the UIV construction, a variable-input-length tweakable cipher realised from a tweakable blockcipher $\widetilde{\mathsf{E}}$ and a VOL-PRF F.

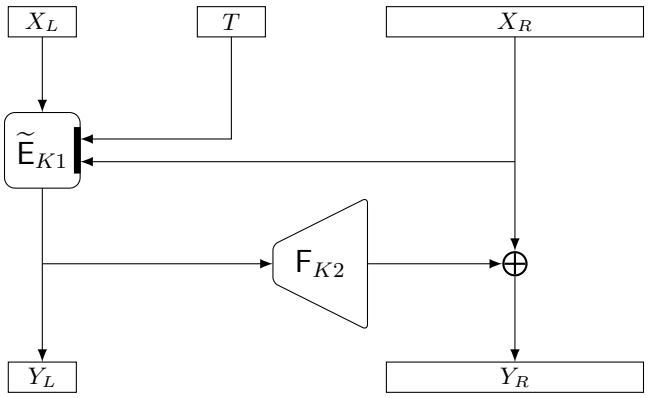

Fig. 4: Graphical representation of the UIV enciphering algorithm.

queries under the constraint that $q_{\mathrm{gu}}v \leq 2^{n-1}$*, there exist adversaries* $\mathcal{B}$ *and* $\mathcal{C}$ *such that*

$$\mathbf{Adv}^{\mathrm{rprp}}_{\mathtt{UIV}}(\mathcal{A}, v) \leq \mathbf{Adv}^{\mathrm{sprp}}_{\widetilde{\mathsf{E}}}(\mathcal{B}) + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{C}) + \frac{q_{\mathrm{gu}}v}{2^{n-1}} + \frac{q_1(q_1-1)}{2^{n+1}} + \frac{q_{\mathrm{en}}(q_{\mathrm{en}}-1)}{2^{n+1}} + \frac{q_2(q_2-1)}{2^{n+m+1}},$$

where $q_1 = q_{\mathrm{en}} + q_{\mathrm{de}} + q_{\mathrm{gu}}$ *and* $q_2 = q_{\mathrm{en}} + q_{\mathrm{de}}$*. The* SPRP *adversary* $\mathcal{B}$ *makes at most* q_{en} *encipher queries and* $q_{\mathrm{de}} + q_{\mathrm{gu}}$ *decipher queries, whereas the* PRF *adversary* $\mathcal{C}$ *makes at most* $q_{\mathrm{en}} + q_{\mathrm{de}} + q_{\mathrm{gu}}$ *queries.*

Concrete UIV Instantiations. We described the UIV construction generically in terms of a fixed-input-length tweakable cipher (FIL-TBC) with variable tweak length and a variable-output-length pseudorandom function (VOL-PRF). The tweakable cipher can be instantiated either via the LRW2 construction using a blockcipher like AES and an Almost XOR-Universal (AXU) hash function like POLYVAL [19]. Alternatively one could use a tweakable blockcipher with a fixed-size tweak, like Deoxys-TBC [23] or SKINNY [4] and augment it with an AXU hash via the XTX transform [24].

As for the variable-output-length PRF it can be instantiated by a blockcipher operated in counter mode. In this case, the tricky part is to match the block size of the FIL-TBC with the input size of the VOL-PRF (equivalent to the IV in the counter mode instantiation). If counter mode uses a blockcipher with a block size equal to the block size of the FIL-TBC then the IV needs to be blinded with an additional key, acting as a universal hash, to avoid colliding counter values.

Notably, UIV can be fully instantiated from AES and POLYVAL which benefits from the native instruction sets present in many modern-day processors. The corresponding instantiation, GCM-UIV, shares many similarities with GCM-SIV [19] (e.g. two-pass enciphering/encryption and one-pass deciphering/decryption) and its performance profile is also similar.

4 Encode-then-Encipher From Rugged PRPs

The encode-then-encipher paradigm is a generic approach, dating back to Bellare and Rogaway [9], for turning a variable-length cipher into an authenticated encryption scheme. Shrimpton and Terashima later extended this paradigm to cater for modern primitives such as tweakable ciphers and nonce-based AEAD [32]. However, both works require that the variable-length cipher satisfy SPRP security for the resulting authenticated encryption scheme to be secure. In this section, we show how to construct nonce-based AEAD from tweakable ciphers that are only RPRP secure. The asymmetric security properties of RPRPs prompt us to consider two schemes with complementary security properties, EtE and EtD, as well as nonce-hiding variants of each.

4.1 Encode-then-Encipher (EtE) Scheme

The first scheme, EtE, achieves *misuse-resistance* (MRAE) security and is the most natural as it uses the encipher algorithm to encrypt and the decipher algorithm to decrypt. It employs a rugged pseudorandom permutation $\widetilde{\mathsf{EE}}$ with domain $\{0,1\}^n \times \{0,1\}^{\geq m}$ and tweak space $\{0,1\}^*$, an injective mapping $\langle\cdot,\cdot\rangle$ from string pairs to single strings, and a function $\mathsf{Func}_2 : \{0,1\}^* \to \{0,1\}^n$. Its pseudocode is presented in

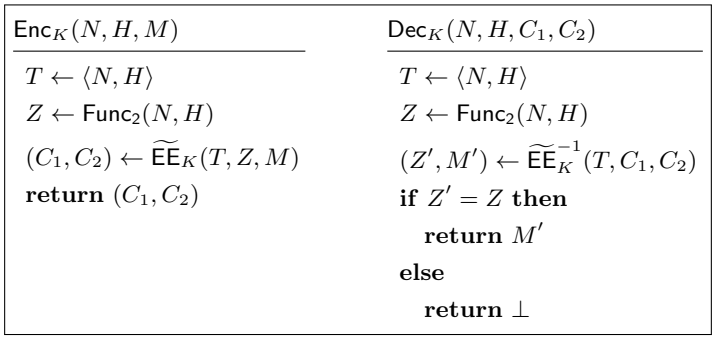

Fig. 5: The EtE construction transforming a variable-length RPRP into a misuse-resistant nonce-based AEAD scheme.

Fig. 5. Note that since C_1 is of fixed size, C_1 and C_2 can be concatenated into a single-string ciphertext to fit the usual AEAD syntax, and any such ciphertext can easily be parsed back into such a pair.

Intuitively, the scheme is misuse resistant since altering any of N, H, or M results in an almost uniformly random ciphertext, by the pseudorandomness of the encipher algorithm. Authenticity is achieved via the function Func_2 under the sole assumption that it be deterministic. Namely, it can be instantiated through a hash function or more simply via truncation (assuming (N, H) is always at least n bits long), or by the constant function (e.g. $\mathsf{Func}_2(N, H) = 0^n$). Then, by RPRP security, it follows that altering either C_1 or C_2 will result in a value of Z' that is unpredictable. Accordingly, the condition $Z' = Z$ will only be satisfied with small probability, irrespective of the specific value of $Z \leftarrow \mathsf{Func}_2(N, H)$. It is worth noting that in reducing the MRAE security of EtE to the RPRP security of $\widetilde{\mathsf{EE}}$, the reduction only makes encipher and guess queries (with $v = 1$), i.e., the decipher oracle is not used at all. This is because in the EtE construction, the verification algorithm can be simulated entirely through the guess oracle. Below is the formal security theorem, the proof of which can be found in Appendix C.1.

Theorem 2. *Let* EtE *be the nonce-based AEAD scheme defined in Fig. 5 realized from a tweakable cipher over the domain* $\{0,1\}^n \times \{0,1\}^{\geq m}$. *Then for any adversary* $\mathcal{A}$ *making* q_e *encryption queries and* q_v *verification queries, there exists an adversary* $\mathcal{B}$ *such that*

$$\mathbf{Adv}^{\mathrm{mrae}}_{\mathtt{EtE}}(\mathcal{A}) \;\leq\; \mathbf{Adv}^{\mathrm{rprp}}_{\widetilde{\mathsf{EE}}}(\mathcal{B}, 1) + \frac{q_e^2}{2^{n+m+1}},$$

where $\mathcal{B}$ *makes* q_e *encipher queries,* q_v *guess queries, and its runtime is similar to that of* $\mathcal{A}$.

4.2 Encode-then-Decipher (EtD) Scheme

In our second scheme, EtD, we switch the roles of the encipher and decipher algorithms, i.e., we decipher to encrypt and encipher to decrypt. By making this switch, we now obtain an AEAD scheme that is secure against the *release of unverified plaintext* (RUPAE). The EtD construction is presented in Fig. 6 together with the associated leakage function used to prove it RUPAE secure. In addition to the variable-length tweakable cipher, the construction makes use of an injective mapping $\langle \cdot, \cdot \rangle$ from string pairs to single strings and a (δ, t)-collision resistant deterministic function Func_3.

The full pseudorandomness of the encipher algorithm, which is now used for decryption, is what makes the scheme RUPAE secure. However, using the decipher algorithm to encrypt presents some new challenges in the security proof due to the constraints in the RPRP security definition. The requirement to never repeat Y_L values across decipher queries is easily satisfied by ensuring that distinct nonces result in distinct Z values. In our generic treatment we fulfill this condition by requiring that the function Func_3 be (δ, t)-collision resistant. On the other hand, the requirement to not forward Y_L values from the encipher oracle to the decipher oracle is a bit more challenging to address in the security proof. Finally, a peculiarity of the EtD construction is that the nonce is included both in the evaluation of Z as well as the tweak, which may seem unnecessary at first. However, its inclusion in the evaluation of Z is necessary to ensure that Y_L values do not repeat as it is the only AEAD input that is guaranteed to be distinct across encryption calls. At the same time its inclusion in the tweak is necessary for RUPAE security, as otherwise the adversary could forward a ciphertext from the encryption oracle to the leakage oracle with a different nonce and, in the real world, recover the original message. The security of EtD is formally stated below in Theorem 3 whose proof can be found in Appendix C.2.

$\mathsf{Enc}_K(N,H,M)$	$\mathsf{Dec}_K(N,H,C_1,C_2)$	$\Lambda_K(N,H,C_1,C_2)$
$T \leftarrow \langle N,H\rangle$	$T \leftarrow \langle N,H\rangle$	$T \leftarrow \langle N,H\rangle$
$Z \leftarrow \mathsf{Func}_3(N,H,M)$	$(Z',M') \leftarrow \widetilde{\mathsf{EE}}_K(T,C_1,C_2)$	$(Z',M') \leftarrow \widetilde{\mathsf{EE}}_K(T,C_1,C_2)$
$(C_1,C_2) \leftarrow \widetilde{\mathsf{EE}}_K^{-1}(T,Z,M)$	$Z \leftarrow \mathsf{Func}_3(N,H,M')$	$Z \leftarrow \mathsf{Func}_3(N,H,M')$
return (C_1,C_2)	**if** $Z' = Z$ **then**	**if** $Z' = Z$ **then**
	return M'	**return** $\top$
	else	**else**
	return $\perp$	**return** M'

Fig. 6: The EtD construction, presented as a subtle AEAD scheme, transforming a variable-length RPRP into a RUPAE-secure AEAD scheme.

Theorem 3. *Let* EtD *be the subtle AEAD scheme defined in Fig. 6 composed from a tweakable cipher over the domain* $\{0,1\}^n \times \{0,1\}^{\geq m}$*, and let* Func_3 *be a* (δ,t)*-collision resistant deterministic function. Then there exists a leakage simulator* S*, such that for any adversary* $\mathcal{A}$ *making* $q_\mathrm{e} \leq 2^{n-1}$ *encryption queries,* q_d *decryption queries,* q_l *queries to the leakage oracle and running in time* t*, there exist* RPRP *adversaries* $\mathcal{B}$ *and* $\mathcal{C}$ *such that*

$$\mathbf{Adv}^{\mathrm{rupae}}_{\mathtt{EtD}}(\mathcal{A},\mathsf{S}) \leq \mathbf{Adv}^{\mathrm{rprp}}_{\widetilde{\mathsf{EE}}}(\mathcal{B},1) + \mathbf{Adv}^{\mathrm{rprp}}_{\widetilde{\mathsf{EE}}}(\mathcal{C},1) + \delta + \frac{(q_\mathrm{e}+1)(q_\mathrm{d}+q_\mathrm{l})}{2^{n-1}} + \frac{(q_\mathrm{e}+q_\mathrm{d}+q_\mathrm{l})^2}{2^{n+m}} + \frac{q_\mathrm{e}(q_\mathrm{d}+q_\mathrm{l})}{2^n}.$$

The adversary $\mathcal{B}$ *makes* q_e *queries to* EN *oracle,* $q_\mathrm{d}+q_\mathrm{l}$ *queries to* DE *oracle, and its runtime is similar to that of* $\mathcal{A}$*. The adversary* $\mathcal{C}$ *makes at most* q_e *queries to* EN *oracle, at most* $q_\mathrm{d}+q_\mathrm{l}$ *queries to* DE *oracle, and its runtime is similar to that of* $\mathcal{A}$*.*

4.3 Nonce-Hiding Variants of EtE and EtD

Up to this point our treatment has focused on the classical nonce-based syntax, but both constructions can be adapted to the nonce-hiding syntax while retaining analogous security properties. Intuitively, the main differences are that encryption now needs to embed the nonce in the ciphertext and the nonce is no longer available during decryption. We describe below how these differences affect each construction.

In the case of EtE, as before the redundancy Z must be located in the left input for security and consequently the nonce has to be embedded in the right input. As the nonce is not available to the decryption algorithm, Z can no longer depend on it. Furthermore Z cannot depend on any value contained in the right part. This is because in the security proof decryption is simulated through the GU oracle, which does not return the right part, and thus the reduction would not be able to evaluate Z. As a result, the possibilities for instantiating the redundancy function are severely restricted here and we simply set $Z = 0^n$ instead.

In the case of EtD, since we are using the decipher algorithm to encrypt the left input must not repeat, and thus this makes it the natural choice of location for embedding the nonce. Accordingly the

$\mathsf{Enc}_K(N,H,M)$	$\mathsf{Dec}_K(H,C_1,C_2)$
$Z \leftarrow 0^n$	$Z \leftarrow 0^n$
$(C_1,C_2) \leftarrow \widetilde{\mathsf{EE}}_K(H,Z,N\|M)$	$(X_L,X_R) \leftarrow \widetilde{\mathsf{EE}}_K^{-1}(H,C_1,C_2)$
return (C_1,C_2)	$N'\|M' \leftarrow X_R$
	if $X_L = Z$ **then**
	return (N',M')
	else
	return $\perp$

Fig. 7: Nonce-hiding variant of the EtE construction. The resulting scheme is MRAE secure if the underlying tweakable cipher is RPRP secure.

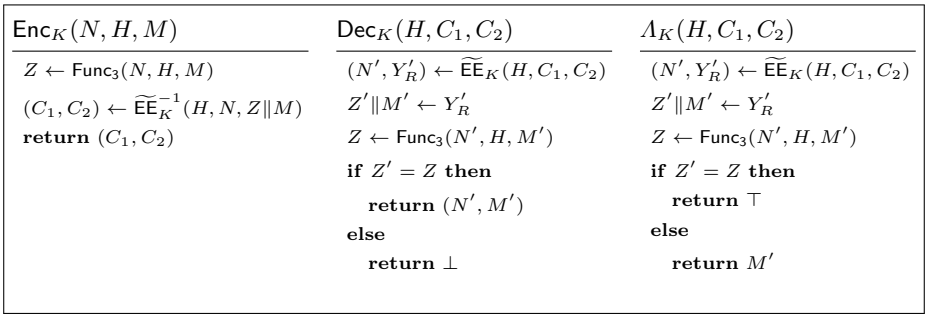

Fig. 8: Nonce-hiding variant of the EtD construction. The resulting scheme is RUPAE secure if the underlying tweakable cipher is RPRP secure.

redundant value Z has to be moved to the right input, which is now possible in the case of EtD since the encipher algorithm is fully pseudorandom and non-malleable. Again, the nonce is not an input to the decryption algorithm, but in this case Z can depend on the nonce as the decryption algorithm can use the nonce that it recovers from the left part. However the nonce still cannot be included in the tweak. Interestingly, the attack that required us to include the nonce in the tweak for EtD is no longer applicable in the nonce-hiding setting, and thus this variant is also RUPAE secure. Note that for this construction Func_3 is only required to be a deterministic function and need not be collision resistant.

Pseudocode descriptions of the nonce-hiding variants of EtE and EtD are given in Fig. 7 and Fig. 8. The security proofs for these variants proceed in a similar fashion to the original nonce-based schemes and we omit them to avoid tedious repetition.

4.4 Instantiations and Related Constructions

Compared to prior works [8, 9, 21, 32], our treatment of the encode-then-encipher paradigm is the first to prove the security of the resulting AEAD by assuming a strictly weaker security notion than SPRP on the part of the cipher. In this light, our results on the MRAE security of EtE and its nonce-hiding variant are analogous to the construction in [32] and the HN5 construction in [8], respectively. Similarly, our result on the RUPAE security of nonce-based EtD is analogous to that in [21] for the closely-related notion of robust AEAD.

For generality, we specified the nonce-based constructions through the redundancy functions Func_2 and Func_3 which can be instantiated in a number of ways. Note that the redundancy functions are generally only required to be deterministic functions, except in nonce-based EtD, which additionally requires Func_3 to be (δ, t)-collision resistant. Thus, one could instantiate these with hash functions or, when applicable, more simply as constant functions that always return 0^n. Clearly, some instantiations are more advantageous in terms of efficiency, while others may prove to be beneficial in extended security models that we did not consider here. Instantiating the nonce-hiding variant of EtD with $\mathsf{Func}_3(N,H,M) := 0^n$ and GCM-UIV recovers the GCM-RUP scheme from [2]. However our treatment exposes other possibilities, such as $\mathsf{Func}_3(N,H,M) := N$, which is trivially $(0,\infty)$-collision resistant. In particular, instantiating the nonce-based variant of EtD with this redundancy function gives rise to a RUPAE-secure AEAD scheme with more compact ciphertexts than GCM-RUP or HN5, as it makes do without the extra n zero bits (assuming the nonce is also n bits long).

When instantiated with $\mathsf{Func}_3(N,H,M) := N$ the nonce-based variants of EtE and EtD will also conceal the nonce, even if decryption does not strictly fit the nonce-hiding syntax. Such a combination of nonce-concealing and compact ciphertexts is beneficial for constructing secure channels over a transmission protocol with out-of-order delivery, like UDP. Indeed, DTLS 1.3 and QUIC go through considerable efforts to achieve this. In Section 6 we will show how this approach, of employing an RPRP and overloading the use of the nonce for both authentication and message indexing, can be used to construct such secure channels more simply and in a more modular fashion. However, we first need to introduce a new type of authenticated encryption that better fits this purpose and allows the receiver to adopt different policies as to how to process ciphertexts that are delivered out of order. In the next section, we present this new and more general type of authenticated encryption and show how it can be realised generically from any nonce-hiding AEAD scheme or directly from an RPRP with the additional benefit of more compact ciphertexts.

5 Nonce-Set AEAD

A secure channel protocol operating over UDP, which may deliver ciphertexts out of order, requires some mechanism to recover the original ordering of the messages. Typically, such secure channels employ an AEAD scheme and overload the nonce to act as the *message number*. Here, nonce-hiding AEAD is advantageous because it attaches the nonce in encrypted form to the ciphertext, thereby making it available to the receiver for recovering the original ordering of messages without leaking the side information contained in the nonce. In Section 4.4 we showed how in the encode-then-encipher paradigm the nonce could be additionally overloaded to act as the redundant bits in the encoding that provide authenticity. This resulted in more compact ciphertexts, but it required that the nonce be already available to the receiver before decryption takes place. Thus, our technique of overloading the nonce for providing authentication is not compatible with a scenario where ciphertexts are delivered out of order, as the receiver is unable to determine the nonce associated with a ciphertext before decrypting it.

In practice, the amount of reordering that takes place over UDP will, on average, be limited. Accordingly, secure channel protocols will typically employ some form of window mechanism which determines which message numbers (and corresponding ciphertexts) can be accepted. If the message number of a ciphertext falls outside the window, it means that the ciphertext is either too old or too far ahead of the ones received and will be discarded. Such window mechanisms can take various forms and can implement a variety of different policies that determine how to deal with replays, when and how to change the window size, and when to advance the window ahead. Nevertheless, at an abstract level, they all specify a limited set of message numbers that can be accepted at that particular point in time.

We propose nonce-set AEAD as a new type of authenticated encryption that lends itself particularly well to this kind of scenario. The main change is that decryption will now additionally take a set of nonces as its input, and for it to succeed, the ciphertext has to be deemed valid with respect to that set of nonces. The motivation for introducing this primitive is twofold. The first is that it will enable the generic construction, which we present in the next section, for a secure channel operating over UDP that can support multiple different window policies. At a very high level, this construction combines a nonce-set AEAD scheme together with a tuple of algorithms that emulate the window mechanism by generating the nonce set for the decryption algorithm and updating it accordingly. This construction is appealing because although the security of the secure channel depends crucially on this tuple of algorithms, it turns out that they only need to satisfy a "functional" requirement and need not at all be cryptographic. In addition, this single construction can be tuned to realize various types of secure-channel behaviour. As such, nonce-set AEAD appears to be the right place for drawing the boundary between cryptographic and non-cryptographic processing. The second and complementary reason for introducing nonce-set AEAD is that we can realize it directly from an RPRP through an encode-then-encipher approach where authentication is achieved by overloading the nonce, thereby yielding more compact ciphertexts. Thus, by introducing nonce-set AEAD, we are now able to simultaneously accommodate these two mechanisms, which were otherwise incompatible. Below is the formal definition.

5.1 Formal Definition

Syntax. A nonce-set encryption scheme $\mathtt{NSE} = (\mathsf{Enc}, \mathsf{Dec})$ is a pair of algorithms with an associated key space $\mathcal{K}$, a nonce space $\mathcal{N} = \{0,1\}^t$ for some $t \in \mathbb{N}$, a nonce-set space $\mathcal{W} \subseteq \mathcal{P}(\mathcal{N})$, a header space $\mathcal{H}$, a message space $\mathcal{M}$ and a ciphertext space $\mathcal{C}$.

- The encryption algorithm follows the usual syntax, i.e.,

$$\mathsf{Enc} : \mathcal{K} \times \mathcal{N} \times \mathcal{H} \times \mathcal{M} \to \mathcal{C}.$$

  As before, encryption must be length-regular, thereby requiring the existence of a function clen, mapping the triple $(|N|, |H|, |M|)$ to the ciphertext length.

- The decryption algorithm works analogously to that in a nonce-hiding encryption scheme but additionally takes a set of nonces $\boldsymbol{W} \in \mathcal{W}$ as part of its input. That is, its syntax is given by

$$\mathsf{Dec} : \mathcal{K} \times \mathcal{W} \times \mathcal{H} \times \mathcal{C} \to (\mathcal{N} \times \mathcal{M}) \cup \{(\perp, \perp)\}.$$

  In addition, for all valid inputs $(K, \boldsymbol{W}, H, C)$ it must hold that:

$$\text{if } \mathsf{Dec}_K(\boldsymbol{W}, H, C) = (N', M') \neq (\perp, \perp) \text{ then } N' \in \boldsymbol{W}.$$

Correctness. For every nonce-set encryption scheme, it must hold that for all $(K, N, H, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{H} \times \mathcal{M}$ and every $\boldsymbol{W} \in \mathcal{W}$ such that $N \in \boldsymbol{W}$,

$$\text{if } C \leftarrow \mathsf{Enc}_K(N, H, M) \text{ then } (N, M) \leftarrow \mathsf{Dec}_K(\boldsymbol{W}, H, C).$$

Security. As before, security requires that no adversary can distinguish the real encryption and decryption algorithms ($\mathsf{Enc}(\cdot,\cdot,\cdot)$, $\mathsf{Dec}(\cdot,\cdot,\cdot)$) from the ideal ones ($\$(\cdot,\cdot,\cdot)$, $\perp(\cdot,\cdot,\cdot)$), under the condition that its encryption queries be nonce-respecting and it does not forward queries from the encryption oracle to the decryption oracle. The main difference to the classical nonce-based AEAD lies in how a forwarding query is defined. This is a query $(\boldsymbol{W}, H, C)$ to the decryption oracle where C was returned in a prior encryption query (N, H, M) and $N \in \boldsymbol{W}$. In other words, the adversary cannot query a ciphertext under a nonce-set containing the nonce with which it was produced. The security of a nonce-set encryption scheme is expressed through the following advantage measure.

Definition 5 (nsAE **Advantage).** *Let* $\mathtt{NSE} = (\mathsf{Enc}, \mathsf{Dec})$ *be a nonce-set based encryption scheme with associated spaces* $(\mathcal{K}, \mathcal{N}, \mathcal{W}, \mathcal{H}, \mathcal{M}, \mathcal{C})$*. Then for any nonce-respecting adversary* $\mathcal{A}$ *that does not make forwarding queries its advantage is defined as*

$$\mathbf{Adv}^{\mathrm{nsae}}_{\mathtt{NSE}}(\mathcal{A}) = \left|\Pr\left[K \leftarrow\!\$\, \mathcal{K} : \mathcal{A}^{\mathsf{Enc}_K(\cdot,\cdot,\cdot),\mathsf{Dec}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} \Rightarrow 1\right]\right|.$$

Misuse Resistance. The above security notion can be strengthened to the misuse-resistance setting in the usual way. Namely by lifting the nonce-respecting requirement and simply requiring that the adversary never query the same triple (N, H, M) to the encryption more than once.

Unpacking the Definition. Note that if we set $\mathcal{W} = \{\{N\} : N \in \mathcal{N}\}$ then nonce-set AEAD effectively reduces to standard nonce-based AEAD with a nonce space $\mathcal{N}$. Thus, nonce-set AEAD can be seen as a natural extension of nonce-based AEAD. Our syntax requires that when decryption succeeds, it associates the decrypted ciphertext to a nonce in $\boldsymbol{W}$. Conversely, this means that if $\boldsymbol{W}$ is the empty set then decryption must fail. In addition, correctness guarantees that when $\boldsymbol{W}$ contains the nonce that was used to produce that ciphertext, decryption will recover the plaintext and will additionally recover that nonce. Finally, besides ruling out forgeries involving new ciphertexts, security also ensures that an adversary is unable to associate an honestly generated ciphertext to a different nonce. These features will come in handy in the next section where we show how to generically transform a nonce-set AEAD into an order-resilient channel.

A practical scheme must specify a format for representing $\boldsymbol{W}$ as a string. In general, this formatting must be concise for the scheme to be efficient. This will, in turn, impose heavy restrictions on the space $\mathcal{W}$ of all possible nonce sets that the decryption algorithm can accept. Thus, an important parameter of a nonce-set AEAD scheme is the maximum nonce set size w, defined as

$$w := \max_{\boldsymbol{W} \in \mathcal{W}} |\boldsymbol{W}|.$$

The specific value of this parameter may be a result of the formatting used to represent $\boldsymbol{W}$, or it may need to be specifically restricted in order to guarantee a certain level of security. In addition, the formatting used to represent $\boldsymbol{W}$ will typically require an efficient means to do membership testing. This aspect of a nonce-set AEAD is beyond our scope. Still, it suffices to say that various instantiations exist that satisfy these requirements, including the formatting used in the window mechanisms employed by existing internet protocols.

5.2 Nonce-Set AEAD From Nonce-Hiding AEAD

Nonce-set AEAD can be easily realized from any nonce-hiding AEAD scheme simply by following decryption with a test verifying that the recovered nonce is in $\boldsymbol{W}$. This construction is shown in Fig. 26, in Appendix D.1. Thus the nonce-hiding constructions by Bellare, Ng, and Tackmann in [8] which are nonce-recovering, namely HN1, HN2, HN4, and HN5, can be readily transformed into nonce-set AEAD schemes. However, these constructions all incur a ciphertext expansion resulting from the underlying integrity mechanism as well as a second ciphertext expansion arising from the nonce encryption. In contrast, the construction we present next reduces this overhead by constructing a nonce-set AEAD scheme directly from a RPRP via the encode-then-encipher paradigm.

$\mathsf{Enc}_K(N, H, M)$	$\mathsf{Dec}_K(\boldsymbol{W}, H, C_1, C_2)$
$l \leftarrow \lvert M \rvert$	$(X_L, X_R) \leftarrow \widetilde{\mathsf{EE}}_K^{-1}(H, C_1, C_2)$
$(C_1, C_2) \leftarrow \widetilde{\mathsf{EE}}_K(H, N \Vert \lfloor M \rfloor_{n-t}, \lceil M \rceil_{l-n+t})$	$N' \leftarrow \lfloor X_L \rfloor_t$
return (C_1, C_2)	$M' \leftarrow \lceil X_L \rceil_{n-t} \Vert X_R$
	if $N' \in \boldsymbol{W}$ **then**
	return (N', M')
	else
	return $(\bot, \bot)$

Fig. 9: The AwN construction, transforming an RPRP-secure cipher $\widetilde{\mathsf{EE}}$ over the domain $\{0,1\}^n \times \{0,1\}^{\geq m}$ into a nonce-set AEAD scheme that is MRAE secure. The scheme is parametrized by the nonce length t, where $t \leq n$.

5.3 The Authenticate-With-Nonce (AwN) Construction

The Authenticate-with-Nonce (AwN) construction is similar in spirit to the EtE construction instantiated with $\mathsf{Func}_2(N, H) = N$, but it gives rise instead to a nonce-set AEAD scheme. A pseudocode description is provided in Fig. 9. Note that the integrity check is now done by verifying that $N' \in \boldsymbol{W}$, rather than an equality test as in EtE. By RPRP security, any mauled ciphertext will produce a left output that is hard to guess, thereby limiting the probability of this condition being satisfied, as long as $\boldsymbol{W}$ is not too large. As a result, the MRAE security of AwN depends on the maximum nonce set size w. Moreover, for added generality, we allow the nonce size t to be smaller or equal to the size of the left domain n. This too bears influence over the security of AwN. In combination, these two aspects give rise to the $w2^{n-t}$ term in the RPRP advantage term within the security bound. The MRAE security of AwN is formally stated in Theorem 4, the proof of which can be found in Appendix D.2.

Theorem 4. *Let* AwN *be the nonce-set AEAD scheme defined in Fig. 9, realized from a tweakable cipher over the domain* $\{0,1\}^n \times \{0,1\}^{\geq m}$*, with nonce size* t *and maximum nonce set size* w*. Then for any* MRAE *adversary* $\mathcal{A}$ *making* q_e *encryption queries and* q_v *verification queries, there exists an* RPRP *adversary* $\mathcal{B}$ *such that*

$$\mathbf{Adv}^{\mathrm{mrae}}_{\mathtt{AwN}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{rprp}}_{\widetilde{\mathsf{EE}}}(\mathcal{B}, w2^{n-t}) + \frac{q_e^2}{2^{n+m+1}}.$$

The adversary $\mathcal{B}$ *makes* q_e *queries to the* EN *oracle and* q_v *queries to the* GU *oracle, and runs in time similar to that of* $\mathcal{A}$*.*

6 Application to Order-Resilient Secure Channels

Equipped with the notion of nonce-set AEAD, we now turn our attention to constructing secure channels over an unreliable transport. QUIC [22, 33] and DTLS [27, 28], which operate over UDP, are two prime examples of order-resilient secure channels. Two recent works [16,17] have analyzed the security of QUIC. Here we will follow in large part the formal security model of Fischlin, Günther, and Janson [17] which builds on and improves over prior works [6, 12, 31] and is the most versatile.

As pointed out already in [12, 13] several strategies are possible for dealing with out-of-order delivery and replay protection. However, their models fail to capture the more elaborate ones that rely on window mechanisms, as in the case of QUIC and DTLS. These window mechanisms can handle out-of-order delivery and replay protection without consuming too much memory and bandwidth. This comes, however, at the expense of added complexity that is harder to model mathematically. Even formulating correctness for such secure channels becomes rather challenging. To overcome the limitations of prior security models, Fischlin et al. replace the level-sets in [31] with a *support predicate*, which essentially serves to determine which ciphertexts should be accepted by the receiver. The point of this predicate is that it considers the receiver's perspective in making this determination. As is the case with QUIC and DTLS a ciphertext deemed invalid at a certain point in time (due to it falling outside the current window) may become valid later (when the window has shifted sufficiently ahead).

Our focus in this section is not to analyze the security of QUIC or DTLS. Instead, we take a fresh perspective on how such secure channels can be constructed differently and more simply through nonce-set AEAD. More specifically, we provide a generic construction for transforming any nonce-set AEAD

scheme into a secure channel parametrized by a support predicate. Notably, the construction works for *any* desired support predicate by employing a quadruple of relatively simple algorithms. In turn, the security of the channel construction relies on the security of the nonce-set AEAD scheme and a mild requirement on the quadruple of algorithms with respect to the support predicate. We tame the complexity in constructions like QUIC and DTLS by introducing modularity into the picture and identifying the role of each component. Here the introduction of nonce-set AEAD plays a key role in glueing the different components together and permitting us to generically express the logic behind the support predicate by processing nonce values. We also strengthen the security definition from [17] to reflect the privacy requirement to conceal message numbers from eavesdroppers. In contrast, in [17] message numbers were required to be transmitted in the clear.

In addition, when the nonce-set AEAD scheme is instantiated with our RPRP-based construction, we end up with a secure channel construction that is competitive in comparison to QUIC and DTLS. To start with, it only requires a single key (instead of two) to process each ciphertext. Our nonce-set AEAD scheme can be realized with GCM components, leading to comparable performance to GCM-SIV and offering misuse-resistance. QUIC only transmits a partial nonce in the ciphertext in order to save bandwidth at the expense of an additional window mechanism to reconstruct it. In contrast, our construction transmits the full nonce, thereby simplifying the processing at the receiver's end, but saves bandwidth nonetheless from its overloaded use of the nonce (within the nonce-set AEAD construction) to provide integrity without a MAC tag.

6.1 Order-Resilient Channels

We start by defining the syntax of order-resilient channels. The definitions below are reproduced from [17] and we do not claim any novelty in them. We do, however, make some alterations in them which we point out along the way.

Definition 6 (Channel Syntax). *A channel consists of a triple of algorithms* $\mathtt{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *with associated spaces* $\mathcal{ST}_S$, $\mathcal{ST}_R$, $\mathcal{MN}$, $\mathcal{A}$, $\mathcal{M}$ *and* $\mathcal{C}$ *such that:*

- $(\mathrm{st}_s, \mathrm{st}_r) \leftarrow\!\!\$\ \mathsf{Init}()$. *The probabilistic initialization algorithm that takes no input and returns an initial sender state* $\mathrm{st}_s \in \mathcal{ST}_S$ *and an initial receiver state* $\mathrm{st}_r \in \mathcal{ST}_R$.
- $(\mathrm{st}'_s, C) \leftarrow\!\!\$\ \mathsf{Send}(\mathrm{st}_s, A, M)$. *The send algorithm, may be probabilistic or stateful and takes as input a sender state* $\mathrm{st}_s \in \mathcal{ST}_S$, *associated data* $A \in \mathcal{A}$ *and a message* $M \in \mathcal{M}$, *and returns as output an updated sender state* $\mathrm{st}'_s \in \mathcal{ST}_S$ *and a ciphertext* $C \in \mathcal{C}$ *or the error symbol* $\perp$.
- $(\mathrm{st}'_r, mn, M) \leftarrow \mathsf{Recv}(\mathrm{st}_r, A, C)$. *The deterministic receive algorithm takes as input a receiver state* $\mathrm{st}_r \in \mathcal{ST}_R$, *associated data* $A \in \mathcal{A}$ *and a ciphertext* $C \in \mathcal{C}$. *It then returns an updated receiver state* $\mathrm{st}'_r \in \mathcal{ST}_R$ *together with, either a message number* $mn \in \mathcal{MN}$ *and a message* $M \in \mathcal{M}$, *or a pair of error symbols* $(\perp, \perp)$.

In comparison to [17] we augmented the Recv algorithm to return the message number together with the message. This reflects the real-world necessity that a higher layer needs such information to correctly position each message in the sequence in which it was sent. We also expanded Send with associated data A, and removed the auxiliary data and accompanying function as they are no longer needed in our setting.

The Support Predicate. There are varying degrees to which a channel may be order resilient. As explained in [17] the prior models by [12,31] are not expressive enough to capture the order resilience of real-world protocols like QUIC and DTLS. To address this, they introduced the support predicate. In essence, the support predicate expresses the channel's tolerance to reordered, replayed, or dropped ciphertexts. It essentially captures the 'character' of the channel, which permeates into every aspect of it— from correctness to robustness (which we explain shortly) to security. Indeed, this conforms with and is reminiscent of the silencing approach in [31], but generalizes it further.

The support predicate takes three inputs: a list $\mathcal{C}_S$ of the sent ciphertexts, a list $\mathcal{DC}_R$ of the received ciphertexts together with a boolean value indicating whether each was deemed supported or not, and a candidate ciphertext C and returns a boolean indicating whether C is supported or not. Thus, whether a ciphertext is supported may depend on the ciphertexts sent, the ones received, and how the current ciphertext relates to them.

$\mathsf{supp}_{\mathrm{rr}[\mathrm{w}_r,\mathrm{w}_f]}(\mathcal{C}_S, \mathcal{DC}_R, C)$

$j \leftarrow \mathrm{index}(C, \mathcal{C}_S)$
$\mathsf{next} \leftarrow \max\{\mathrm{index}(C, \mathcal{C}_S) : (\mathbf{true}, C) \in \mathcal{DC}_R\} + 1$
if $j = \perp$ **then** // $C \notin \mathcal{C}_S$
  return false
if $(j < \mathsf{next} - \mathrm{w}_r) \vee (j > \mathsf{next} + \mathrm{w}_f)$ **then** // outside of replay/reordering window
  return false
if $(\mathbf{true}, C) \in \mathcal{DC}_R$ **then** // replay
  return false
else
  return true

Fig. 10: Example support predicate $\mathsf{supp}_{\mathrm{rr}[\mathrm{w}_r,\mathrm{w}_f]}$ corresponding to a channel functionality with replay protection and a sliding window of size $(\mathrm{w}_r + \mathrm{w}_f)$.

In conformance with [17], any ciphertext not in $\mathcal{C}_S$ must not be supported. However, whereas in [17] the list $\mathcal{C}_S$ is allowed to contain repeating ciphertexts, we specifically prohibit this. In particular, we require that every entry in $\mathcal{C}_S$ be identified uniquely. Whether two messages encrypt to the same ciphertext (a possibility with stateful schemes) or not depends on the scheme at hand. Thus allowing this to occur would render the support predicate scheme-specific, thereby introducing a circularity in the correctness and security definitions—which is why we avoid this possibility. Moreover, the representation of ciphertexts should not bear any weight on the predicate's value. Therefore, we allow ciphertexts to be identified by integers or other strings as long as the entries in $\mathcal{C}_S$ are unique.

There are two other minor points where we deviate from [17]. One is that we require that every support predicate accept perfectly-in-order delivery. The other is that we allow the support predicate to only return a boolean value, whereas in the formulation in [17] it could also return an integer. This seems to have been required due to the possibility of repeating ciphertexts in $\mathcal{C}_S$, which we specifically rule out.

An example support predicate is specified in Fig. 10, reflecting the required functionality of a typical real-world protocol. First, the submitted ciphertext is checked against the list of sent ciphertexts $\mathcal{C}_S$, and its corresponding location is assigned to j. If no match is found, it returns **false**. The variable next represents the next ciphertext number the receiver expects (calculated as the largest index of a supported ciphertext that was received plus one). Then the ciphertext is supported if it falls within the window centred on next and has not already been received.

Channel Correctness. Different support predicates identify different channel functionalities. Nevertheless, we can define channel correctness generically for *any* possible support predicate. Intuitively, correctness requires that for any supported (and thus honestly generated) ciphertext, the receiver must always be able to recover the original message contained in that ciphertext together with its corresponding message number. Thus correctness ensures that the receiver is able to recover the original sequence of messages in the exact ordering in which they were sent. This is formally defined via the game in Fig. 11.

Definition 7 (Channel Correctness). *A channel* $\mathtt{Ch}$ *is said to be correct with respect to a support predicate* supp*, if for all possible adversaries* $\mathcal{A}$ *it holds that*

$$\Pr\left[\mathsf{CORR}^{\mathcal{A}}_{\mathtt{Ch},\mathsf{supp}} \Rightarrow 1\right] = 0.$$

6.2 The Robustness Property

Unlike TLS and similar protocols, where one invalid ciphertext typically results in the connection being torn down, order-resilient channels are inherently required to tolerate a significant amount of decryption failures during their operation. Such decryption failures may arise from the unreliable nature of the underlying protocol, or due to manipulation by a malicious adversary. Furthermore, the receiver will generally be unable to distinguish between these two cases. Thus, order-resilient channels must maintain their correct operation in the presence of adversarial manipulation. However, the above correctness requirement

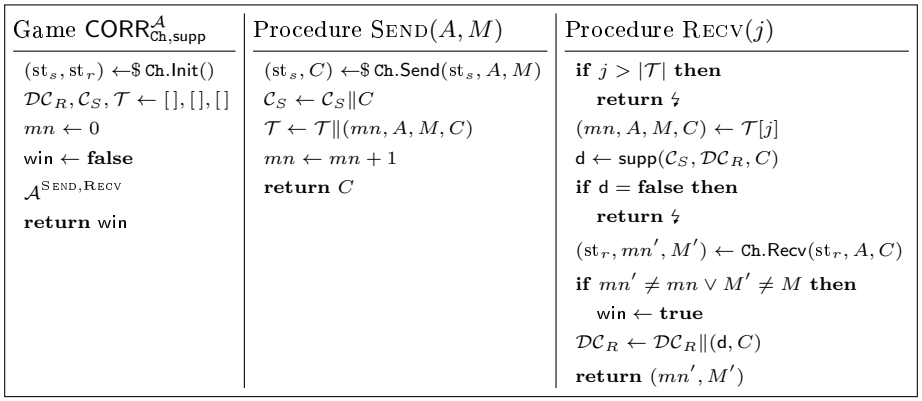

Fig. 11: The game CORR used to define channel correctness.

does not capture such a scenario as it considers only honestly-generated ciphertexts. Accordingly, [17] introduced the notion of *robustness* to capture this stronger requirement.

Robustness is formally defined through the ROB game described in Fig. 29, in Appendix E.1. Here, the Recv oracle maintains internally two Recv instances, the *real* one, which is supplied with all queried ciphertexts, and the *correct* one, which is only supplied with supported ciphertexts. Then if at any point the adversary queries a supported ciphertext that causes the outputs of the two Recv instances to differ, it will constitute a win for the adversary. The advantage of an adversary is quantified as its probability of winning this game.

Definition 8 (ROB Advantage). *For a channel* $\mathtt{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *and a support predicate* supp*, the corresponding robustness advantage of an adversary* $\mathcal{A}$ *is defined as:*

$$\mathbf{Adv}^{\mathrm{rob}}_{\mathtt{Ch},\mathsf{supp}}(\mathcal{A}) = \Pr\left[\mathsf{ROB}^{\mathcal{A}}_{\mathtt{Ch},\mathsf{supp}} \Rightarrow 1\right].$$

Note that in the ROB game both Recv instances are initialized with the same state. Thus, for the adversary to win, the states of the two instances must at some point diverge. On the other hand, only unsupported ciphertexts can cause such a divergence in their states. Therefore, a sufficient condition for satisfying robustness is that unsupported ciphertext do not change the state.

6.3 Channel Security

We use a single-game definition of channel security that combines confidentiality and integrity into one notion. It is heavily based on the security definitions from [17], without robustness, and adapted with some of the ideas from simulatable channels in [14]. Security is defined via the indistinguishability game INT-SIM-CCA shown in Fig. 12. Here, we essentially require the existence of a stateless algorithm S that can simulate the Send oracle to the adversary and that the adversary is unable to query an unsupported ciphertext to Recv that decrypts successfully, i.e., a forgery. Note that, as shown in [14], requiring the simulator S to be stateless results in a stronger security notion. Namely, it provides key privacy and ensures that ciphertexts do not leak the message number since the simulator cannot keep track of the number of messages that are sent. Below is the formal definition.

Definition 9 (INT-SIM-CCA Advantage). *Let* $\mathtt{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a channel protocol realizing the functionality corresponding to the support predicate* supp*. Then,* Ch *is* INT-SIM-CCA *secure if there exists a stateless encryption simulator* S *such that for any adversary* $\mathcal{A}$ *the following quantity is small*

$$\mathbf{Adv}^{\text{int-sim-cca}}_{\mathtt{Ch},\mathsf{supp}}(\mathcal{A}, \mathsf{S}) = \left|2\Pr\left[\mathsf{INT\text{-}SIM\text{-}CCA}^{\mathcal{A},\mathsf{S}}_{\mathtt{Ch},\mathsf{supp}} \Rightarrow 1\right] - 1\right|.$$

6.4 From Nonce-Set AEAD to Order-Resilient Secure Channels

We are now ready to present this section's main contribution - a generic construction for transforming any nonce-set AEAD scheme into an order-resilient channel. This construction consists of a nonce-set

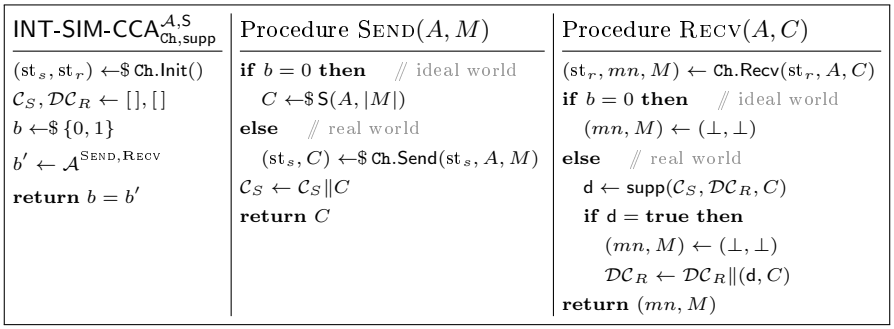

Fig. 12: The INT-SIM-CCA game used to define channel security.

AEAD scheme combined with a tuple of four basic algorithms called the *nonce-set processing scheme* algorithms. This construction has some notable features. Firstly, it works for *any* support predicate. This means that this template construction can be used to realize any channel functionality that can be expressed via the support predicate introduced by Fischlin et al. in [17]. In addition, any instantiation will automatically satisfy robustness and channel security for that support predicate. The main conditions for this to hold are that the underlying nonce-set AEAD be secure and that the nonce-set processing scheme *faithfully* reproduce the functionality of the support predicate.

As the name implies, the nonce-set processing scheme algorithms are primarily concerned with generating and updating the nonce-set that is fed to the nonce-set AEAD. The faithfulness property ensures that the nonce-set processing scheme accurately reflects the channel behaviour corresponding to the support predicate. Recall that we required the support predicate to be defined over any possible way of identifying the ciphertexts as long as it uniquely represented each ciphertext in $\mathcal{C}_S$. This means that we can identify each ciphertext with the nonce it is assigned in the Send algorithm. Accordingly, the role of the nonce-set processing algorithms is to identify the set of supported nonces at every stage of the Recv algorithm. Our channel construction will then use the set of supported nonces as the nonce set to be fed to the nonce-set AEAD. Thus our generic construction can be viewed as decomposing a channel into these constituent components, thereby adding to our understanding of order-resilient channels.

We start by describing the syntax of the nonce-set processing scheme algorithms. A nonce-set processing scheme NSP consists of the following constituent algorithms:

- $(\mathrm{st}_s, \mathrm{st}_r) \leftarrow\$ \mathsf{StInit}()$. A probabilistic initialization algorithm, that returns the initial sender state st_s and the initial receiver state st_r.
- $(\mathrm{st}'_s, N) \leftarrow \mathsf{NonceExtract}(\mathrm{st}_s)$. A deterministic nonce extraction algorithm, that takes as input the non-key component of the sender state and returns a (possibly) updated state together with a *unique* nonce N or the symbol $\perp$.
- $\boldsymbol{W} \leftarrow \mathsf{NonceSetPolicy}(\mathrm{st}_r)$. A deterministic nonce-set policy algorithm that takes as input the non-key component of the receiver state and returns a nonce set.
- $(\mathrm{st}'_r, mn) \leftarrow \mathsf{StUpdate}(\mathrm{st}_r, N)$. A deterministic state-update algorithm that takes as input the non-key component of the receiver state together with a nonce, and returns an updated state together with the message number corresponding to that nonce.

The Faithfulness Property. The only property that we require from a nonce-set processing scheme is that it *faithfully* reproduces the functionality of the channel's support predicate. Note that none of the nonce-set processing scheme algorithms makes use of a secret key. This is because faithfulness is a property that can be satisfied without cryptographic means. For any scheme, NSP and support predicate supp, faithfulness is defined via the game FAITHFUL shown in Fig. 13. The adversary's goal is to cause the nonce-set processing algorithms and the support predicate to be misaligned or recover the wrong message number from a nonce. Note that the receiver is only allowed to query nonces to the F-Recv oracle that the F-Send oracle has returned. A win occurs if the submitted nonce is supported, but not contained in the nonce set returned by NonceSetPolicy or the message number returned by StUpdate for that nonce is incorrect. Alternatively, if the nonce is not supported but the nonce set does contain that nonce, it is also a win for the adversary.

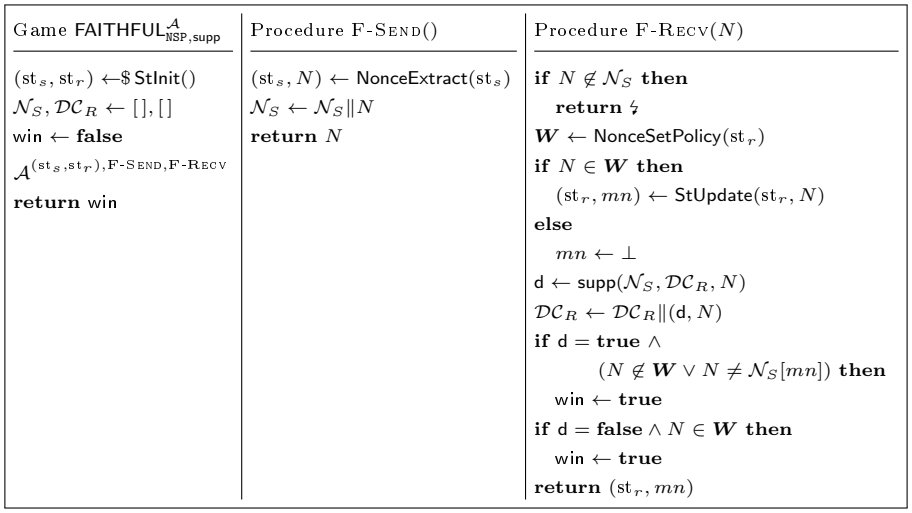

Fig. 13: The game FAITHFUL used to define faithfulness for a tuple of nonce-set processing algorithms.

Definition 10 (FAITHFUL Advantage). *Let* NSP *be a nonce-set processing scheme. Then for any adversary $\mathcal{A}$ and any support predicate* supp*, the corresponding advantage is defined as*

$$\mathbf{Adv}^{\text{faithful}}_{\mathtt{NSP},\mathsf{supp}}(\mathcal{A}) = \Pr\left[\mathsf{FAITHFUL}^{\mathcal{A}}_{\mathtt{NSP},\mathsf{supp}} \Rightarrow 1\right].$$

We say that a nonce-set scheme NSP faithfully reproduces the support predicate supp, if for all possible adversaries $\mathcal{A}$ it holds that $\mathbf{Adv}^{\text{faithful}}_{\mathtt{NSP},\mathsf{supp}}(\mathcal{A}) = 0$.

Generic Channel Construction. Our generic construction of an order-resilient secure channel $\mathtt{Ch}_{\mathrm{NS}} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ from a nonce-set AEAD scheme $\mathtt{NSE} = (\mathsf{Enc}, \mathsf{Dec})$ and a nonce-set processing scheme $\mathtt{NSP} = (\mathsf{StInit}, \mathsf{NonceExtract}, \mathsf{NonceSetPolicy}, \mathsf{StUpdate})$ is presented in Fig. 14.

Channel Correctness. The proof of correctness for this generic construction is provided in Appendix E.2.

Theorem 5. *If the nonce-set AEAD scheme* NSE *is correct and the nonce-set processing scheme* NSP *faithfully reproduces the support predicate* supp*, then the channel construction* $\mathtt{Ch}_{\mathrm{NS}}$ *presented in Fig. 14 is correct with respect to* supp*.*

Channel Robustness. We argue robustness based on our earlier observation that a sufficient condition for robustness is that unsupported ciphertexts never affect the channel state. The faithfulness of NSP guarantees that only the nonces used to generate supported ciphertexts will be included in the nonce set. Then, by the nsAE security of NSE, decryption can only succeed as long as the ciphertext was produced by the sender under one of the nonces contained in the nonce set—otherwise, it would constitute a forgery. Thus decryption will always fail for unsupported ciphertexts, and by construction, the state is never updated (StUpdate is not called) when decryption fails.

Channel Security. The security of $\mathtt{Ch}_{\mathrm{NS}}$ is formally stated in the following theorem whose proof can be found in Appendix E.3.

Theorem 6 (Security of $\mathtt{Ch}_{\mathrm{NS}}$). *Let* $\mathtt{Ch}_{\mathrm{NS}}$ *be the generic channel construction described in Fig. 14, composed from a nonce-set AEAD scheme* NSE *with associated ciphertext space* $\{0,1\}^{\geq \ell}$ *and a nonce-set processing scheme* NSP*. Then, for any support predicate* supp *there exists a stateless simulator* S*, such that for every* INT-SIM-CCA *adversary $\mathcal{A}$ making q_{s} send queries and q_{r} receive queries, there exist adversaries $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\mathbf{Adv}^{\text{int-sim-cca}}_{\mathtt{Ch}_{\mathrm{NS}},\mathsf{supp}}(\mathcal{A},\mathsf{S}) \;\leq \mathbf{Adv}^{\text{nsae}}_{\mathtt{NSE}}(\mathcal{B}) + \mathbf{Adv}^{\text{faithful}}_{\mathtt{NSP},\mathsf{supp}}(\mathcal{C}) + \frac{q_{\mathrm{s}}(q_{\mathrm{s}}-1)}{2^{\ell}}.$$

Furthermore, $\mathcal{B}$ makes q_{s} encryption queries and at most q_{r} decryption queries, whereas $\mathcal{C}$ makes q_{s} send queries and at most q_{r} receive queries. Both adversaries run in time similar to that of $\mathcal{A}$.

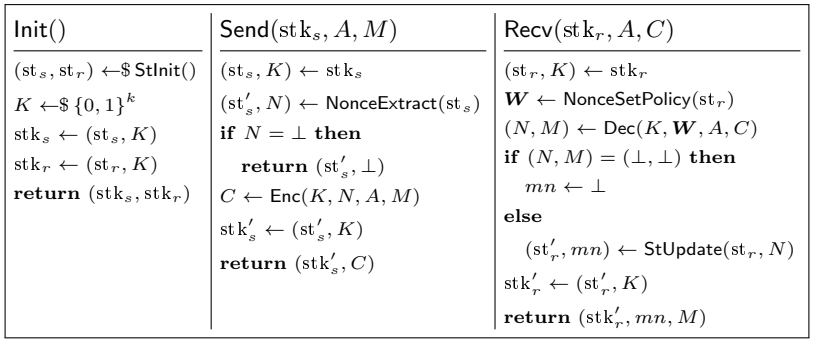

Fig. 14: A generic construction of an order-resilient secure channel $\mathtt{Ch}_{\mathrm{NS}}$ from a nonce-set AEAD scheme and a nonce-set processing scheme.

Concrete Nonce-Set Processing Scheme. In Appendix E.4 we present a concrete realization of NSP that faithfully reproduces the example support predicate $\mathsf{supp}_{\mathrm{rr}[\mathrm{w}_r,\mathrm{w}_f]}$ described in Fig. 10.

Acknowledgments

We are grateful to Alessandro Melloni, Jean-Pierre Münch, and Martijn Stam for their input regarding the RPRP security definition and other helpful discussions. We also thank the anonymous CRYPTO 2022 reviewers for their thorough reading and constructive comments.

References

1. E. Andreeva, A. Bogdanov, A. Luykx, B. Mennink, N. Mouha, and K. Yasuda. How to securely release unverified plaintext in authenticated encryption. In P. Sarkar and T. Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 105–125. Springer, Heidelberg, Dec. 2014.
2. T. Ashur, O. Dunkelman, and A. Luykx. Boosting authenticated encryption robustness with minimal modifications. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 3–33. Springer, Heidelberg, Aug. 2017.
3. G. Barwell, D. Page, and M. Stam. Rogue decryption failures: Reconciling AE robustness notions. In Groth [18], pages 94–111.
4. C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim. SKINNY-AEAD and SKINNY-hash. *IACR Trans. Symm. Cryptol.*, 2020(S1):88–131, 2020.
5. M. Bellare and S. Keelveedhi. Authenticated and misuse-resistant encryption of key-dependent data. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 610–629. Springer, Heidelberg, Aug. 2011.
6. M. Bellare, T. Kohno, and C. Namprempre. Authenticated encryption in SSH: Provably fixing the SSH binary packet protocol. In V. Atluri, editor, *ACM CCS 2002*, pages 1–11. ACM Press, Nov. 2002.
7. M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Okamoto [26], pages 531–545.
8. M. Bellare, R. Ng, and B. Tackmann. Nonces are noticed: AEAD revisited. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 235–265. Springer, Heidelberg, Aug. 2019.
9. M. Bellare and P. Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In Okamoto [26], pages 317–330.
10. D. J. Bernstein. *CAESAR competition call for submissions*, 2014. https://competitions.cr.yp.to/caesar-call.html.
11. A. Boldyreva, J. P. Degabriele, K. G. Paterson, and M. Stam. On symmetric encryption with distinguishable decryption failures. In S. Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 367–390. Springer, Heidelberg, Mar. 2014.
12. C. Boyd, B. Hale, S. F. Mjølsnes, and D. Stebila. From stateless to stateful: Generic authentication and authenticated encryption constructions with application to TLS. In K. Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 55–71. Springer, Heidelberg, Feb. / Mar. 2016.
13. J. Chan and P. Rogaway. Anonymous AE. In S. D. Galbraith and S. Moriai, editors, *ASIACRYPT 2019, Part II*, volume 11922 of *LNCS*, pages 183–208. Springer, Heidelberg, Dec. 2019.
14. J. P. Degabriele and M. Fischlin. Simulatable channels: Extended security that is universally composable and easier to prove. In T. Peyrin and S. Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 519–550. Springer, Heidelberg, Dec. 2018.

15. J. P. Degabriele, V. Karadžić, A. Melloni, J.-P. Münch, and M. Stam. Rugged pseudorandom permutations and their applications. Presented at the IACR Real World Crypto Symposium 2022.
16. A. Delignat-Lavaud, C. Fournet, B. Parno, J. Protzenko, T. Ramananandro, J. Bosamiya, J. Lallemand, I. Rakotonirina, and Y. Zhou. A security model and fully verified implementation for the IETF QUIC record layer. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2021.
17. M. Fischlin, F. Günther, and C. 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.
18. J. Groth, editor. *15th IMA International Conference on Cryptography and Coding*, volume 9496 of *LNCS*. Springer, Heidelberg, Dec. 2015.
19. S. Gueron and Y. Lindell. GCM-SIV: Full nonce misuse-resistant authenticated encryption at under one cycle per byte. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015*, pages 109–119. ACM Press, Oct. 2015.
20. S. Halevi and P. Rogaway. A parallelizable enciphering mode. In T. Okamoto, editor, *CT-RSA 2004*, volume 2964 of *LNCS*, pages 292–304. Springer, Heidelberg, Feb. 2004.
21. V. T. Hoang, T. Krovetz, and P. Rogaway. Robust authenticated-encryption AEZ and the problem that it solves. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 15–44. Springer, Heidelberg, Apr. 2015.
22. J. Iyengar and M. Thomson. *RFC9000: QUIC: A UDP-Based Multiplexed and Secure Transport*, 2021. https://datatracker.ietf.org/doc/html/rfc9000.
23. J. Jean, I. Nikolic, T. Peyrin, and Y. Seurin. The Deoxys AEAD family. *Journal of Cryptology*, 34(3):31, July 2021.
24. K. Minematsu and T. Iwata. Tweak-length extension for tweakable blockciphers. In Groth [18], pages 77–93.
25. C. Namprempre, P. Rogaway, and T. Shrimpton. Reconsidering generic composition. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 257–274. Springer, Heidelberg, May 2014.
26. T. Okamoto, editor. *ASIACRYPT 2000*, volume 1976 of *LNCS*. Springer, Heidelberg, Dec. 2000.
27. E. Rescorla and N. Modadugu. *The Datagram Transport Layer Security Version 1.2*, 2012. https://datatracker.ietf.org/doc/html/rfc6347.
28. E. Rescorla, H. Tschofenig, and N. Modadugu. *The Datagram Transport Layer Security (DTLS) Protocol Version 1.3 IETF Draft*, 2021. https://datatracker.ietf.org/doc/draft-ietf-tls-dtls13/.
29. P. Rogaway. Nonce-based symmetric encryption. In B. K. Roy and W. Meier, editors, *FSE 2004*, volume 3017 of *LNCS*, pages 348–359. Springer, Heidelberg, Feb. 2004.
30. P. Rogaway and T. Shrimpton. A provable-security treatment of the key-wrap problem. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 373–390. Springer, Heidelberg, May / June 2006.
31. P. Rogaway and Y. Zhang. Simplifying game-based definitions - indistinguishability up to correctness and its application to stateful AE. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 3–32. Springer, Heidelberg, Aug. 2018.
32. T. Shrimpton and R. S. Terashima. A modular framework for building variable-input-length tweakable ciphers. In K. Sako and P. Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 405–423. Springer, Heidelberg, Dec. 2013.
33. M. Thomson and S. Turner. *RFC9001: Using TLS to Secure QUIC*, 2021. https://datatracker.ietf.org/doc/html/rfc9001.

Appendix

A Appendix Section 2

A.1 SPRP Security

A tweakable cipher is typically required to satisfy one of two security notions, of which we only present the stronger one that we use in this work. This is the SPRP notion[2], which requires that for a uniformly sampled key K the tweakable cipher be indistinguishable from an ideal cipher $\widetilde{\Pi}$ sampled uniformly from $\mathsf{IC}(\mathcal{T}, \mathcal{X})$. The corresponding advantage term is formally defined below.

Definition 11 (SPRP Advantage). *Let $\widetilde{\mathsf{EE}}$ be a tweakable cipher defined over $(\mathcal{K}, \mathcal{T}, \mathcal{X})$. Then for any adversary $\mathcal{A}$ its* SPRP *advantage is defined as:*

$$\mathbf{Adv}^{\mathrm{sprp}}_{\widetilde{\mathsf{EE}}}(\mathcal{A}) = \left|\Pr\left[K \leftarrow\!\$\, \mathcal{K} : \mathcal{A}^{\widetilde{\mathsf{EE}}_K(\cdot,\cdot),\widetilde{\mathsf{EE}}_K^{-1}(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\widetilde{\Pi} \leftarrow\!\$\, \mathsf{IC}(\mathcal{T}, \mathcal{X}) : \mathcal{A}^{\widetilde{\Pi}(\cdot,\cdot),\widetilde{\Pi}^{-1}(\cdot,\cdot)} \Rightarrow 1\right]\right|$$

A.2 Nonce-Hiding AEAD

Nonce-hiding encryption was recently introduced in [8] and is essentially a variant of nonce-based encryption in which the nonce is embedded in encrypted form in the ciphertext and then recovered upon decryption. Thus, this alternative syntax aims to achieve better privacy by encrypting the nonce. Most of the security notions for nonce-based encryption can be straightforwardly adapted to this syntax, but this slight difference in syntax automatically gives rise to slightly different security properties. For the sake of brevity, we will only outline the changes one needs to make to adapt the prior syntax and security definitions of nonce-based encryption to nonce-hiding encryption.

Syntax. A nonce-hiding encryption scheme $\mathtt{NHE} = (\mathsf{Enc}, \mathsf{Dec})$ is a pair of algorithms were encryption behaves exactly as in a nonce-based encryption scheme but the decryption algorithm operates as follows:

$$\mathsf{Dec} : \mathcal{K} \times \mathcal{H} \times \mathcal{C} \to (\mathcal{N} \times \mathcal{M}) \cup \{\perp\}\,.$$

Note that now decryption does not take a nonce as part of its input but instead it returns it as part of its output. Prior works [2, 8] did not generally require decryption to return the nonce as part of its output. In [8] a nonce-recovering formulation was considered only as a special case through the addition of an extra algorithm specifically intended for this purpose. In contrast, we require the decryption algorithm to always return the nonce so that the corresponding security definitions will guarantee that it is authenticated together with the other outputs. In practice, the nonce is typically used by the higher-level protocols to recover the correct ordering of messages and possibly other metadata. It is therefore crucial that decryption returns it and authenticates it.

A nonce-hiding subtle encryption scheme is defined analogously by simply adapting the decryption algorithm to the syntax above and simply dropping the nonce from the input to the leakage function, that is:

$$\Lambda : \mathcal{K} \times \mathcal{H} \times \mathcal{C} \to \{0,1\}^* \cup \{\top\}\,.$$

Security. The nAE, MRAE and RUPAE security definitions can be directly adapted to the nonce-hiding setting by adjusting the syntax of the decryption, verification, leakage, and simulator algorithms in the natural way. In each of the three games, the restriction on forwarding queries is now redefined to mean that the adversary cannot make a query of the form (H, C) if it previously queried (N, H, M), for some $N \in \mathcal{N}$, and got C in return.

A.3 Difference Lemma

In the proof of Theorem 3 (`EtD` construction), presented in Appendix C.2, we make use of the following difference lemma.

[2] More precisely, this is the *tweakable* SPRP notion, often denoted as STPRP or $\widetilde{\mathrm{SPRP}}$, but as we do not use the non-tweaked variants here we simply denote it as SPRP.

Claim. For events A_0, A_1, E_0 and E_1 it holds that:

$$|\Pr[A_0] - \Pr[A_1]| \leq |\Pr[A_0 \wedge \neg E_0] - \Pr[A_1 \wedge \neg E_1]| + \max\{\Pr[E_0], \Pr[E_1]\}.$$

Proof. Starting from the left hand side we introduce events E_0 and E_1 to obtain

$$|\Pr[A_0] - \Pr[A_1]| = |\Pr[(A_0 \wedge E_0) \vee (A_0 \wedge \neg E_0)] - \Pr[(A_1 \wedge E_1) \vee (A_1 \wedge \neg E_1)]|\,.$$

Bounding compound events and applying the union bound yields

$$\leq |\Pr[A_0 \wedge \neg E_0] + \Pr[E_0] - \Pr[A_1 \wedge \neg E_1] + \Pr[E_1]|.$$

We can remove the modulus from the above expression to restate it as

$$\begin{aligned} = \max\{&\Pr[A_0 \wedge \neg E_0] + \Pr[E_0] - \Pr[A_1 \wedge \neg E_1], \\ &\Pr[A_1 \wedge \neg E_1] + \Pr[E_1] - \Pr[A_0 \wedge \neg E_0]\} \end{aligned}$$

Rearranging terms and collecting them yields

$$\leq |\Pr[A_0 \wedge \neg E_0] - \Pr[A_1 \wedge \neg E_1]| + \max\{\Pr[E_0], \Pr[E_1]\}. \qquad \square$$

B Appendix Section 3

B.1 RRND Security

In order to prove our UIV construction RPRP secure, we will use a related notion called RRND security. We proceed by first defining and explaining what RRND security is, and then present the Lemma 2 that relates RPRP and RRND security. We use exactly this lemma later in Appendix B.2 to prove UIV construction is a secure RPRP.

In the case of RRND security, an adversary has access to the same set of oracles as in the RPRP game. However, it is tasked with distinguishing $\widetilde{\mathsf{EE}}$ from a tweakable two-sided random function $\widetilde{\mathsf{RR}}$ instead of an ideal cipher. The tweakable two-sided random function is a function sampled uniformly at random from the set $\pm\mathrm{Func}(\mathcal{T}, \mathcal{X}_L \times \mathcal{X}_R)$. In turn, this can be viewed as a family of random functions from $\mathcal{X}_L \times \mathcal{X}_R$ to $\mathcal{X}_L \times \mathcal{X}_R$ indexed by $\{+,-\} \times \mathcal{T}$ where each tweak in $\mathcal{T}$ identifies a forward $(+)$ function and a backward $(-)$ function. Note that with high probability, the forward and backward functions will not be inverses of each other, which would allow the adversary to easily distinguish $\widetilde{\mathsf{EE}}$ from $\widetilde{\mathsf{RR}}$. Accordingly, in addition to the restrictions present in the RPRP game, we also prohibit the adversary from forwarding queries from De to En. This is enforced via the set $\mathcal{P}$. The game defining RRND security is given in Fig. 15, and the corresponding advantage formula is given below.

Definition 12 (RRND Advantage). *Let $\widetilde{\mathsf{EE}}$ be a tweakable cipher over a split domain $(\mathcal{X}_L \times \mathcal{X}_R)$. Then for a positive integer v and an adversary $\mathcal{A}$ attacking the* RRND *security of $\widetilde{\mathsf{EE}}$ the corresponding advantage is defined as*

$$\mathbf{Adv}^{\mathrm{rrnd}}_{\widetilde{\mathsf{EE}}}(\mathcal{A}, v) = \left|2\Pr\left[\mathsf{RRND}^{\mathcal{A},v}_{\widetilde{\mathsf{EE}}} \Rightarrow 1\right] - 1\right|.$$

In [20] Halevi and Rogaway prove a strengthened analogue of the switching lemma stating that for sufficiently large domains an ideal cipher and a tweakable two-sided random function are indistinguishable. This result is reproduced in the following lemma.

Lemma 1 (Lemma 6. in [20]). *Let $\mathcal{A}$ be an adversary that does not forward the result of one oracle to another. Then it holds*

$$\left|\Pr\left[\widetilde{\Pi} \leftarrow\!\$\, \mathsf{IC}(\mathcal{T}, \mathcal{X}) : \mathcal{A}^{\widetilde{\Pi}(\cdot,\cdot), \widetilde{\Pi}^{-1}(\cdot,\cdot)}\right] - \Pr\left[\widetilde{\mathsf{RR}} \leftarrow\!\$\, \pm\mathrm{Func}(\mathcal{T}, \mathcal{X}) : \mathcal{A}^{\widetilde{\mathsf{RR}}(+,\cdot,\cdot), \widetilde{\mathsf{RR}}(-,\cdot,\cdot)}\right]\right| \leq \frac{q(q-1)}{2^{b+1}},$$

where q is the total number of $\mathcal{A}$'s queries to both oracles and b is the length of the shortest element in $\mathcal{X}$.

It then follows, by the above lemma, that RRND security implies RPRP security. Intuitively, the result follows since the main difference between the two corresponding games is that in the ideal world, we replace an ideal cipher for a tweakable two-sided random function. This is stated formally in the lemma below.

Game $\mathsf{RRND}^{\mathcal{A},v}_{\widetilde{\mathsf{EE}}}$	$\textsc{De}(T, Y_L, Y_R)$
$K \leftarrow\!\$\, \mathcal{K}$	**if** $Y_L \in \mathcal{F} \cup \mathcal{R}$
$b \leftarrow\!\$\, \{0,1\}$	**return** $\lightning$
$\mathcal{F}, \mathcal{R}, \mathcal{U}, \mathcal{P} \leftarrow \emptyset, \emptyset, \emptyset, \emptyset$	**if** $b = 0$
$\widetilde{\mathsf{RR}} \leftarrow\!\$\, \pm\mathrm{Func}(\mathcal{T}, \mathcal{X}_L \times \mathcal{X}_R)$	$(X_L, X_R) \leftarrow \widetilde{\mathsf{RR}}(-, T, Y_L, Y_R)$
$b' \leftarrow \mathcal{A}^{\textsc{En},\textsc{De},\textsc{Gu}}$	**else**
return $b = b'$	$(X_L, X_R) \leftarrow \widetilde{\mathsf{EE}}_K^{-1}(T, Y_L, Y_R)$
	$\mathcal{R} \xleftarrow{\cup} \{Y_L\}$; $\mathcal{U} \xleftarrow{\cup} \{(T, Y_L, Y_R)\}$
	$\mathcal{P} \xleftarrow{\cup} \{(T, X_L, X_R)\}$
	return (X_L, X_R)
$\textsc{En}(T, X_L, X_R)$	$\textsc{Gu}(T, Y_L, Y_R, \boldsymbol{V})$
if $(T, X_L, X_R) \in \mathcal{P}$	**if** $((T, Y_L, Y_R) \in \mathcal{U}) \vee (\lvert\boldsymbol{V}\rvert > v)$
return $\lightning$	**return** $\lightning$
if $b = 0$	**if** $b = 0$
$(Y_L, Y_R) \leftarrow \widetilde{\mathsf{RR}}(+, T, X_L, X_R)$	**return false**
else	**else**
$(Y_L, Y_R) \leftarrow \widetilde{\mathsf{EE}}_K(T, X_L, X_R)$	$(X_L, X_R) \leftarrow \widetilde{\mathsf{EE}}_K^{-1}(T, Y_L, Y_R)$
$\mathcal{F} \xleftarrow{\cup} \{Y_L\}$; $\mathcal{U} \xleftarrow{\cup} \{(T, Y_L, Y_R)\}$	**return** $X_L \in \boldsymbol{V}$
return (Y_L, Y_R)	

Fig. 15: The game used to define RRND security for a tweakable cipher $\widetilde{\mathsf{EE}}$.

Lemma 2. *Let $\widetilde{\mathsf{EE}}$ be a tweakable cipher over a split domain $(\mathcal{X}_L \times \mathcal{X}_R)$ where $\mathcal{X}_L \subseteq \{0,1\}^{\geq n}$ and $\mathcal{X}_R \subseteq \{0,1\}^{\geq m}$. Then for a positive integer v and an adversary $\mathcal{A}$ making q_{en} encipher oracle queries, q_{de} decipher oracle queries and q_{gu} guess oracle queries, it holds that*

$$\mathbf{Adv}^{\mathrm{rprp}}_{\widetilde{\mathsf{EE}}}(\mathcal{A}, v) \leq \mathbf{Adv}^{\mathrm{rrnd}}_{\widetilde{\mathsf{EE}}}(\mathcal{A}, v) + \frac{q(q-1)}{2^{n+m+1}},$$

where $q = q_{\mathrm{en}} + q_{\mathrm{de}}$.

Proof. We prove the lemma by using the game-hopping technique. Without loss of generality, we assume the adversary $\mathcal{A}$ does not repeat a query to the En or Gu oracle, nor does it forward De oracle query result to En oracle.

$\mathbf{G}_0$: This is the RPRP game with the bit b set to 0 (ideal world).

$\mathbf{G}_1$: We replace the ideal cipher $\widetilde{\Pi}$ with a tweakable two-sided random function $\widetilde{\mathsf{RR}}$ to obtain the game $\mathbf{G}_1$, given in Fig. 16.

We bound $\mathcal{A}$'s advantage in distinguishing the games $\mathbf{G}_0$ and $\mathbf{G}_1$ by applying the Lemma 1. Let $\mathcal{D}$ be an adversary trying to distinguish between the pairs of oracles $(\widetilde{\Pi}(\cdot,\cdot), \widetilde{\Pi}^{-1}(\cdot,\cdot))$ and $(\widetilde{\mathsf{RR}}(+,\cdot,\cdot), \widetilde{\mathsf{RR}}(-,\cdot,\cdot))$ appearing in Lemma 1. We call $\widetilde{\Pi}(\cdot,\cdot)$ and $\widetilde{\mathsf{RR}}(+,\cdot,\cdot)$ forward oracles, and $\widetilde{\Pi}^{-1}(\cdot,\cdot)$ and $\widetilde{\mathsf{RR}}(-,\cdot,\cdot)$ backward oracles. The adversary $\mathcal{D}$ runs $\mathcal{A}$ and simulates the games $\mathbf{G}_0$ and $\mathbf{G}_1$ for $\mathcal{A}$ using its forward and backward oracles.

In order to simulate the En oracle, the adversary $\mathcal{D}$ uses its forward oracle to query (T, X_L, X_R) and get the result (Y_L, Y_R), which it forwards back to $\mathcal{A}$. We note that querying (T, X_L, X_R) to $\mathcal{D}$'s forward oracle means the tweak T is the first input and the pair (X_L, X_R) is the second input to the oracle[3]. In the case of simulation of the De oracle, the adversary $\mathcal{D}$ uses its backward oracle to query (T, Y_L, Y_R) and get the result (X_L, X_R), which it forwards back to $\mathcal{A}$. As for the Gu oracle simulation, the adversary $\mathcal{D}$ does not need to ask its forward or backward oracle since it is supposed always to return $\perp$ to $\mathcal{A}$.

[3] Same principle applies to queries to the $\mathcal{D}$'s backward oracle.

Game $\mathbf{G}_1$	$\textsc{De}(T, Y_L, Y_R)$
$K \leftarrow\!\$\, \mathcal{K}$	**if** $Y_L \in \mathcal{F} \cup \mathcal{R}$
$\mathcal{F}, \mathcal{R}, \mathcal{U} \leftarrow \emptyset, \emptyset, \emptyset$	**return** $\lightning$
$b \leftarrow \mathcal{A}^{\textsc{En},\textsc{De},\textsc{Gu}}$	$(X_L, X_R) \leftarrow \widetilde{\mathsf{RR}}(-, Y_L, Y_R)$
return b	$\mathcal{R} \xleftarrow{\cup} \{Y_L\}$; $\mathcal{U} \xleftarrow{\cup} \{(T, Y_L, Y_R)\}$
	return (X_L, X_R)
$\textsc{En}(T, X_L, X_R)$	$\textsc{Gu}(T, Y_L, Y_R, \boldsymbol{V})$
$(Y_L, Y_R) \leftarrow \widetilde{\mathsf{RR}}(+, X_L, X_R)$	**if** $((T, Y_L, Y_R) \in \mathcal{U}) \vee (\lvert\boldsymbol{V}\rvert > v)$
$\mathcal{F} \xleftarrow{\cup} \{Y_L\}$; $\mathcal{U} \xleftarrow{\cup} \{(T, Y_L, Y_R)\}$	**return** $\lightning$
return (Y_L, Y_R)	**return false**

Game $\mathbf{G}_2$	$\textsc{De}(T, Y_L, Y_R)$
$K \leftarrow\!\$\, \mathcal{K}$	**if** $Y_L \in \mathcal{F} \cup \mathcal{R}$
$\mathcal{F}, \mathcal{R}, \mathcal{U} \leftarrow \emptyset, \emptyset, \emptyset$	**return** $\lightning$
$b \leftarrow \mathcal{A}^{\textsc{En},\textsc{De},\textsc{Gu}}$	$(X_L, X_R) \leftarrow \widetilde{\mathsf{EE}}^{-1}(K, Y_L, Y_R)$
return b	$\mathcal{R} \xleftarrow{\cup} \{Y_L\}$; $\mathcal{U} \xleftarrow{\cup} \{(T, Y_L, Y_R)\}$
	return (X_L, X_R)
$\textsc{En}(T, X_L, X_R)$	$\textsc{Gu}(T, Y_L, Y_R, \boldsymbol{V})$
$(Y_L, Y_R) \leftarrow \widetilde{\mathsf{EE}}(K, X_L, X_R)$	**if** $((T, Y_L, Y_R) \in \mathcal{U}) \vee (\lvert\boldsymbol{V}\rvert > v)$
$\mathcal{F} \xleftarrow{\cup} \{Y_L\}$; $\mathcal{U} \xleftarrow{\cup} \{(T, Y_L, Y_R)\}$	**return** $\lightning$
return (Y_L, Y_R)	$(X_L, X_R) \leftarrow \widetilde{\mathsf{EE}}^{-1}(K, Y_L, Y_R)$
	return $X_L \in \boldsymbol{V}$

Fig. 16: The games $\mathbf{G}_1$ and $\mathbf{G}_2$ for the proof of Lemma 2.

During the whole time, the adversary $\mathcal{D}$ keeps track of the restrictions on queries made by $\mathcal{A}$ by keeping track of sets $\mathcal{F}, \mathcal{R}$ and $\mathcal{U}$ appearing in games $\mathbf{G}_0$ and $\mathbf{G}_1$.

The adversary $\mathcal{D}$ simulates the games $\mathbf{G}_0$ and $\mathbf{G}_1$ for $\mathcal{A}$ correctly. Utilizing the bound from Lemma 1, it holds that

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_0} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_1} \Rightarrow 1\right]\right| \leq q(q-1)/2^{n+m+1}, \tag{1}$$

since the length of the shortest element in $\mathcal{X}_L \times \mathcal{X}_R$ is at least $n + m$. The number q is the number of $\mathcal{A}$'s queries to En and De oracles.

$\mathbf{G}_2$: This is the RPRP game with the bit b set to 1 (real world). It is given in Fig. 16. We claim the advantage of the adversary $\mathcal{A}$ distinguishing between the games $\mathbf{G}_1$ and $\mathbf{G}_2$ is bounded by its RRND advantage. Indeed, the games $\mathbf{G}_1$ and $\mathbf{G}_2$ correspond to the ideal and real world of the RRND game. Notice how the additional restriction appearing in En oracle in the RRND game (in comparison to $\mathbf{G}_1$ and $\mathbf{G}_2$) is "satisfied" by the assumption that the adversary $\mathcal{A}$ does not forward queries from De to En oracle. Therefore,

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_1} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_2} \Rightarrow 1\right]\right| \leq \mathbf{Adv}^{\mathrm{rrnd}}_{\widetilde{\mathsf{EE}}}(\mathcal{A}, v). \tag{2}$$

Combining inequalities (1) and (2) one achieves the bound in the lemma statement. □

B.2 Proof of Theorem 1 (UIV Construction)

Proof. We structure the proof as follows. First, we show in the following claim that the UIV construction is RRND secure, and then we utilize the result of Lemma 2 to arrive at the RPRP security bound from

the theorem statement. Without loss of generality, we assume the adversary $\mathcal{A}$ does not repeat a query to the En or Gu oracle.

Claim. For a positive integer v and an adversary $\mathcal{A}$ making q_{en} encipher queries, q_{de} decipher queries and q_{gu} guess queries under the constraint that $q_{\text{gu}}v \leq 2^{n-1}$, there exist adversaries $\mathcal{B}$ and $\mathcal{C}$ such that

$$\mathbf{Adv}^{\text{rrnd}}_{\texttt{UIV}}(\mathcal{A}, v) \leq \mathbf{Adv}^{\text{sprp}}_{\widetilde{\mathsf{E}}}(\mathcal{B}) + \mathbf{Adv}^{\text{prf}}_{\mathsf{F}}(\mathcal{C}) + \frac{q_{\text{gu}}v}{2^{n-1}} + \frac{q_1(q_1-1)}{2^{n+1}} + \frac{q_{\text{en}}(q_{\text{en}}-1)}{2^{n+1}},$$

where $q_1 = q_{\text{en}} + q_{\text{de}} + q_{\text{gu}}$. The SPRP adversary $\mathcal{B}$ makes at most q_{en} encipher queries and $q_{\text{de}} + q_{\text{gu}}$ decipher queries, whereas the PRF adversary $\mathcal{C}$ makes at most q queries.

Proof (of the Claim). We prove the claim by using the game-hopping technique.

$\mathbf{G}_0$: This is the RRND game with the bit set to 1 (real world).

$\mathbf{G}_1$: In the game $\mathbf{G}_1$, we replace the tweakable blockcipher $\widetilde{\mathsf{E}}$ with an ideal blockcipher $\widetilde{\Pi} \leftarrow\!\$\, \mathsf{IC}(\mathcal{T} \times \mathcal{X}_R, \mathcal{X}_L)$. We bound $\mathcal{A}$'s distinguishing advantage with the SPRP advantage of $\widetilde{\mathsf{E}}$ by constructing an SPRP adversary $\mathcal{B}$. The adversary $\mathcal{B}$ starts by running $\mathcal{A}$ and simulating the games $\mathbf{G}_0$ and $\mathbf{G}_1$ for it. In order to simulate the calls to the tweakable blockciper in the $\texttt{UIV}$ construction, $\mathcal{B}$ uses its own oracles from the SPRP game. As for the PRF, $\mathcal{B}$ samples a random key $K2$ and calculates $\mathsf{F}_{K2}(\cdot,\cdot)$ by itself. In addition, the adversary $\mathcal{B}$ keeps track of all the sets appearing in the games $\mathbf{G}_0$ and $\mathbf{G}_1$ and enforces the corresponding game rules. In the end, the adversary $\mathcal{B}$ returns the same bit that $\mathcal{A}$ returns.

The adversary $\mathcal{B}$ simulates the games correctly. If it is interacting with the real world of the SPRP game, it simulates the game $\mathbf{G}_0$ to $\mathcal{A}$. Otherwise it is interacting with the ideal world of the SPRP game and it simulates the game $\mathbf{G}_1$ to $\mathcal{A}$. It then holds that

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_0} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_1} \Rightarrow 1\right]\right| \leq \mathbf{Adv}^{\text{sprp}}_{\widetilde{\mathsf{E}}}(\mathcal{B}). \tag{3}$$

$\mathbf{G}_2$: In the game $\mathbf{G}_2$, we replace the VOL-PRF F with a truly random VOL function R^{∞}. We bound $\mathcal{A}$'s distinguishing advantage with the PRF advantage of F by constructing a PRF adversary $\mathcal{C}$. The adversary $\mathcal{C}$ starts by running $\mathcal{A}$ and simulating the games $\mathbf{G}_1$ and $\mathbf{G}_2$ for it. In order to simulate the ideal blockcipher in the construction, the adversary $\mathcal{C}$ lazily samples it and calculates $\widetilde{\Pi}(\cdot,\cdot)$ and $\widetilde{\Pi}^{-1}(\cdot,\cdot)$ by itself. As for the VOL-PRF, $\mathcal{C}$ uses its own oracles from the PRF game to simulate the PRF calls in the $\texttt{UIV}$ construction. In addition, the adversary $\mathcal{C}$ keeps track of all the sets appearing in the games $\mathbf{G}_1$ and $\mathbf{G}_2$, and enforces the corresponding game rules. In the end, the adversary $\mathcal{C}$ returns the same bit that $\mathcal{A}$ returns. The adversary $\mathcal{C}$ simulates the games correctly. If it is interacting with the real world of the PRF game, it simulates the game $\mathbf{G}_1$ to $\mathcal{A}$. Otherwise it is interacting with the ideal world of the PRF game and it simulates the game $\mathbf{G}_2$ to $\mathcal{A}$. Hence, it holds that

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_1} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_2} \Rightarrow 1\right]\right| \leq \mathbf{Adv}^{\text{prf}}_{\mathsf{F}}(\mathcal{C}). \tag{4}$$

$\mathbf{G}_3$: In the game $\mathbf{G}_3$, we change the guess oracle so that it always returns **false**. Towards this goal, we first change the game $\mathbf{G}_2$ to $\mathbf{G}_2^*$, by rewriting the Gu oracle code and introducing the flag bad_{Gu}. The games $\mathbf{G}_2$ and $\mathbf{G}_2^*$ are given in Fig. 17, and they are equivalent. In $\mathbf{G}_3$ one then removes the boxed line if bad_{Gu} is set to **true**, thus leaving the guess oracle in $\mathbf{G}_3$ always returning **false**. It is seen from Fig. 17 that the games $\mathbf{G}_2^*$ and $\mathbf{G}_3$ are identical-until-bad games, so we can bound $\mathcal{A}$'s distinguishing advantage between them by using the fundamental lemma of game playing.

In order to calculate $\Pr[\mathsf{bad}_{\text{Gu}}]$, we first introduce intermediate events $\mathsf{bad}_{\text{Gu},i}$, where the event $\mathsf{bad}_{\text{Gu},i}$ is the event that the flag bad_{Gu} was triggered in $\mathcal{A}$'s i-th query to the guess oracle. Because the tuple (T, Y_L, Y_R) in i-th guess oracle query $(T, Y_L, Y_R, \boldsymbol{V})$ is unused, we can be sure En and De oracles did not leak any information about X_L. If that tuple (T, Y_L, Y_R) is queried to Gu for the first time, $\Pr[\mathsf{bad}_{\text{Gu},i}] = v/2^n$. If it is repeated, $\Pr[\mathsf{bad}_{\text{Gu},i}] \leq v/(2^n - (i-1)v)$ since every time $\mathcal{A}$ makes a repeated query, v number of possibilities for X_L are removed. Therefore, it holds that

$$\begin{aligned}\left|\Pr\left[\mathcal{A}^{\mathbf{G}_2} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_3} \Rightarrow 1\right]\right| \leq \Pr[\mathsf{bad}_{\text{Gu}}] &\leq \sum_{i=1}^{q_{\text{gu}}} \Pr[\mathsf{bad}_{\text{Gu},i}] \leq \sum_{i=1}^{q_{\text{gu}}} \frac{v}{2^n - (i-1)v} \\ &\leq \frac{q_{\text{gu}}v}{2^n - q_{\text{gu}}v} \leq \frac{q_{\text{gu}}v}{2^{n-1}},\end{aligned} \tag{5}$$

under the assumption that $q_{\text{gu}}v \leq 2^{n-1}$.

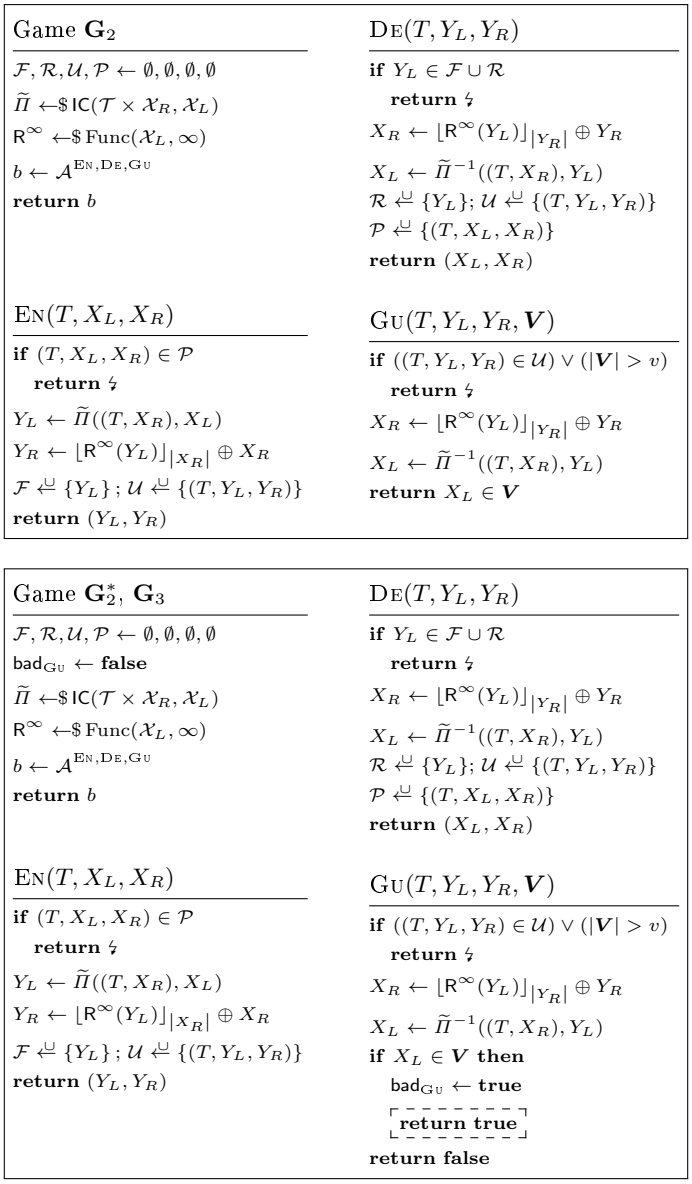

Fig. 17: The games $\mathbf{G}_2$, $\mathbf{G}_2^*$ and $\mathbf{G}_3$ for the proof of Theorem 1 (`UIV` construction). $\mathbf{G}_3$ does not contain the boxed code.

$\mathbf{G}_4$: In the game $\mathbf{G}_4$, we replace the ideal cipher $\widetilde{\Pi}$ with a tweakable two-sided random function $\widetilde{\mathsf{R}}$. We bound $\mathcal{A}$'s distinguishing advantage between the games $\mathbf{G}_3$ and $\mathbf{G}_4$ by utilizing Lemma 1. We construct an adversary $\mathcal{D}$ distinguishing between oracle pairs $(\widetilde{\Pi}(\cdot,\cdot), \widetilde{\Pi}^{-1}(\cdot,\cdot))$ or $(\widetilde{\mathsf{R}}(+,\cdot,\cdot), \widetilde{\mathsf{R}}(-,\cdot,\cdot))$, that will run $\mathcal{A}$ and simulate the games $\mathbf{G}_3$ and $\mathbf{G}_4$ for $\mathcal{A}$. We call $\widetilde{\Pi}(\cdot,\cdot)$ and $\widetilde{\mathsf{R}}(+,\cdot,\cdot)$ forward oracles, and $\widetilde{\Pi}^{-1}(\cdot,\cdot)$ and $\widetilde{\mathsf{R}}(-,\cdot,\cdot)$ backward oracles. If $\mathcal{D}$ is interacting with oracle pair $(\widetilde{\Pi}(\cdot,\cdot), \widetilde{\Pi}^{-1}(\cdot,\cdot))$, it simulates the game $\mathbf{G}_3$ for $\mathcal{A}$. Otherwise, it simulates the game $\mathbf{G}_4$ for $\mathcal{A}$. During the simulation, $\mathcal{D}$ will keep track of the sets $\mathcal{F}, \mathcal{R}, \mathcal{U}$, and $\mathcal{P}$ while exercising the restrictions on the queries imposed by these sets as it is done in $\mathbf{G}_3$ and $\mathbf{G}_4$. The adversary $\mathcal{D}$ will simulate the random VOL function R^∞ appearing in the construction by lazy sampling the needed results. The random function R^∞ needs to be sampled in a consistent way, meaning that for any input string X and any two output lengths l_1 and l_2, with $l_1 \leq l_2$, $\lfloor \mathsf{R}^\infty(X) \rfloor_{l_1}$ needs to be a prefix of $\lfloor \mathsf{R}^\infty(X) \rfloor_{l_2}$. The lazy sampling works in this case as follows. The first time l-bit output of $\mathsf{R}^\infty(X)$ is needed, for some string X, the adversary $\mathcal{D}$ samples a random string of length l and stores it into the function table for input X. Next time, if a l'-bit output of $\mathsf{R}^\infty(X)$ is needed, with $l' \leq l$, the adversary $\mathcal{D}$ reads the stored value, truncates it to l' bits and outputs the truncated string. Otherwise, if $l' > l$, $\mathcal{D}$ reads the l-bit string stored in the function table, extends it

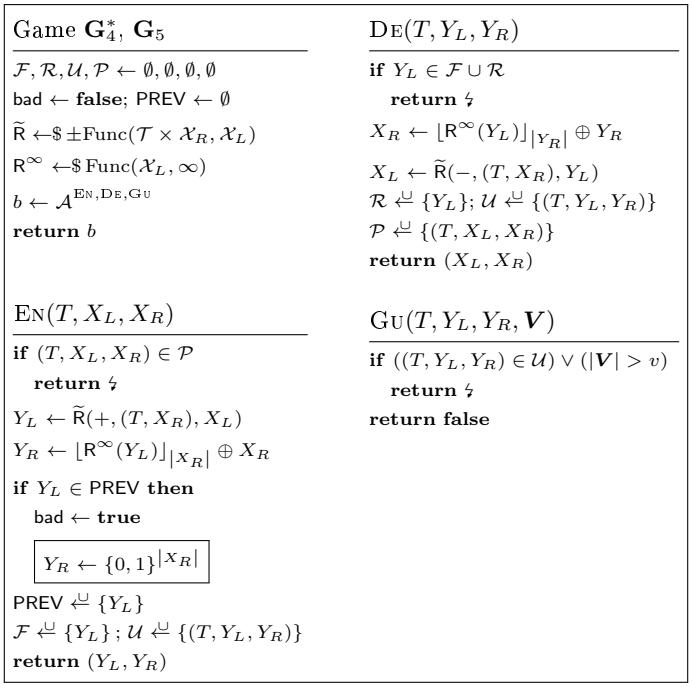

Fig. 18: The games $\mathbf{G}_4^*$ and $\mathbf{G}_5$ for the proof of Theorem 1 (UIV construction). $\mathbf{G}_4$ does not contain the boxed code, while $\mathbf{G}_5$ does.

with new random bits up to length l', stores this l'-bit string back into the table and outputs those l' bits. We continue with describing the adversary $\mathcal{D}$.

It simulates the En oracle as follows. First, it queries its forward oracle with the pair $((T, X_R), X_L)$ to receive the result Y_L and then it calculates Y_R by using the lazily-sampled R^∞. After that, it returns (Y_L, Y_R) back to $\mathcal{A}$. The adversary $\mathcal{D}$ simulates the De oracle by calculating X_R using the lazily-sampled R^∞, then querying its backward oracle with the pair $((T, X_R), Y_L)$ to receive the result X_L. It returns (X_L, X_R) back to $\mathcal{A}$. As for the Gu oracle, the adversary $\mathcal{D}$ simply returns **false** to $\mathcal{A}$. Adversary $\mathcal{D}$ correctly simulates the games for $\mathcal{A}$.

It is important to note that in no case will adversary $\mathcal{D}$ forward a query either from its forward oracle to its backward oracle, or the other way around, thus allowing us to utilize the Lemma 1. Therefore, it holds that

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_3} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_4} \Rightarrow 1\right]\right| \leq \frac{q_1(q_1-1)}{2^{n+1}} \tag{6}$$

$\mathbf{G}_5$: This game is equivalent to the RRND game with the bit set to 0 (ideal world) and it is given in Fig. 18. In order to arrive at the game $\mathbf{G}_5$, we first construct the game $\mathbf{G}_4^*$ from $\mathbf{G}_4$ by adding the flag bad and introducing the set PREV. In addition, in the enciphering oracle, we add the check if Y_L is in the set PREV and the line where bad is set to **true**. The games $\mathbf{G}_4$ and $\mathbf{G}_4^*$ are equivalent. Finally, we switch from the game $\mathbf{G}_4^*$ to the game $\mathbf{G}_5$ by introducing the boxed line of code. As seen in the figure, the games $\mathbf{G}_4$ and $\mathbf{G}_5$ are identical-until-bad games. Therefore, we again apply the fundamental lemma of game playing. It holds that

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_4} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_5} \Rightarrow 1\right]\right| \leq \Pr[\mathsf{bad}] \leq \frac{q_{\mathrm{en}}(q_{\mathrm{en}}-1)}{2^{n+1}}, \tag{7}$$

where the probability of the event bad is the probability of a collision in the output of $\widetilde{\mathsf{R}}(+,\cdot,\cdot)$ happening. Since the triples queried (T, X_L, X_R) to En are always unique, the output of $\widetilde{\mathsf{R}}$ will always be uniformly random and Pr[bad] reduces to a birthday bound.

The only thing left for us is to show that the game $\mathbf{G}_5$ is equivalent to the ideal world of the RRND game. We compare the behaviour of all oracles in $\mathbf{G}_5$ with the behavior of their counterparts in the ideal world of the RRND game.

Queries to En: By the assumption that the adversary does not repeat queries, the query triple (T, X_L, X_R) is always unique. Therefore Y_L is always uniformly random. As for the right output Y_R, there are two

possibilities. In the first case, Y_L has not been appeared before and that makes $\lfloor \mathsf{R}^{\infty}(Y_L) \rfloor_{|X_R|}$ uniformly random. It follows that Y_R will then also be uniformly random. In the other case, Y_R will be sampled uniformly at random. Thus, the output (Y_L, Y_R) will always be uniformly random in the $\mathbf{G}_5$, same as it happens in the ideal world of the RRND game.

Queries to DE: For a query triple (T, Y_L, Y_R), the game(s) restrict that Y_L is not repeated to DE and that it has not been output by EN oracle. It follows that the output X_R will always be uniformly random. From the same fact, it follows that the output X_L will always be uniformly random, since at least the input Y_L to $\widetilde{\mathsf{R}}(-,\cdot,\cdot)$ is always unique.

Queries to GU: The oracle always returns false in both games.

Following the analysis above, we conclude that the adversary $\mathcal{A}$ cannot distinguish between the game $\mathbf{G}_5$ and the ideal world of the RRND game since the corresponding oracles in the two games behave the same.

By combining inequalities (3), (4), (5), (6) and (7) we achieve the claimed bound for RRND security of the `UIV` construction. Together with the result of Lemma 2, the bound from the theorem statement for RPRP security of `UIV` holds as well. □

C Appendix Section 4

C.1 Proof of Theorem 2 (`EtE` Construction)

Proof. We prove the MRAE security of the `EtE` construction by using the following sequence of games.

$\mathbf{G}_0$: This is the real world of the MRAE game and it is given in Fig. 19.

$\mathbf{G}_1$: In the game $\mathbf{G}_1$, we replace the tweakable cipher $\widetilde{\mathsf{EE}}$ with a lazily-sampled ideal cipher $\widetilde{\Pi}$ in the encryption oracle and change the VER oracle always to return $\perp$. The game $\mathbf{G}_1$ is given in Fig. 20.

For any adversary $\mathcal{A}$ distinguishing between the games $\mathbf{G}_0$ and $\mathbf{G}_1$ we construct an RPRP adversary $\mathcal{B}$, that runs $\mathcal{A}$ and simulates $\mathbf{G}_0$ and $\mathbf{G}_1$ for $\mathcal{A}$. The adversary $\mathcal{B}$ works as follows. On $\mathcal{A}$'s encryption oracle query (N, H, M), the adversary $\mathcal{B}$ first calculates T and Z itself and forwards the triple (T, Z, M) to its own enciphering oracle. Upon receiving the result, it forwards it back to $\mathcal{A}$ and updates the set $\mathcal{C}$. On $\mathcal{A}$'s verification oracle query (N, H, C_1, C_2), $\mathcal{B}$ tests whether this query is in $\mathcal{C}$ and returns $\lightning$ if it is. Otherwise, $\mathcal{B}$ goes on to construct T and Z. Finally, the adversary $\mathcal{B}$ makes a guess oracle query $(T, C_1, C_2, \{Z\})$. If GU oracle returns **true** $\mathcal{B}$ returns $\top$ back to $\mathcal{A}$, else $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$.

If $\mathcal{B}$ interacts with the real world of the RPRP game, it simulates the game $\mathbf{G}_0$ to $\mathcal{A}$. Otherwise, it simulates the game $\mathbf{G}_1$. We note that the verification oracle behaviour in the game $\mathbf{G}_0$ corresponds to the guess oracle code in the real world of RPRP game. From the above, it follows that

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_0} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_1} \Rightarrow 1\right]\right| \leq \mathbf{Adv}^{\mathrm{rprp}}_{\widetilde{\mathsf{EE}}}(\mathcal{B}, 1). \tag{8}$$

Game $\mathbf{G}_0$	Procedure $\textsc{Enc}(N, H, M)$	Procedure $\textsc{Ver}(N, H, C_1, C_2)$
$K \leftarrow\!\!\$\, \mathcal{K}$	$T \leftarrow \langle N, H \rangle$	**if** $(N, H, C_1, C_2) \in \mathcal{C}$ **then**
$\mathcal{C} \leftarrow \emptyset$	$Z \leftarrow \mathsf{Func}_2(N, H)$	return $\lightning$
$b' \leftarrow \mathcal{A}^{\textsc{Enc},\textsc{Ver}}$	$(C_1, C_2) \leftarrow \widetilde{\mathsf{EE}}_K(T, Z, M)$	$T \leftarrow \langle N, H \rangle$
return b'	$\mathcal{C} \leftarrow \mathcal{C} \cup \{(N, H, C_1, C_2)\}$	$Z \leftarrow \mathsf{Func}_2(N, H)$
	return (C_1, C_2)	$(Z', M') \leftarrow \widetilde{\mathsf{EE}}_K^{-1}(T, C_1, C_2)$
		$v \leftarrow \perp$
		if $Z' = Z$ **then**
		$v \leftarrow \top$
		return v

Fig. 19: The game $\mathbf{G}_0$ for the proof of Theorem 2 (`EtE` scheme).

Game $\mathbf{G}_1$, $\mathbf{G}_2$	Procedure $\textsc{Enc}(N, H, M)$	Procedure $\textsc{Ver}(N, H, C_1, C_2)$
$K \leftarrow\!\$\, \mathcal{K}$	$T \leftarrow \langle N, H \rangle$	**if** $(N, H, C_1, C_2) \in \mathcal{C}$ **then**
$\mathcal{C} \leftarrow \emptyset$	$Z \leftarrow \mathsf{Func}_2(N, H)$	return $\lightning$
$b' \leftarrow \mathcal{A}^{\textsc{Enc},\textsc{Ver}}$	**if** $\widetilde{\Pi}[T, Z, M] = \perp$ **then**	return $\perp$
return b'	$C_1 \| C_2 \leftarrow\!\$\, \{0,1\}^{n+\lvert M\rvert}$	
	if $\widetilde{\Pi}^{-1}[T, C_1, C_2] \neq \perp$ **then**	
	$\mathsf{bad} \leftarrow$ **true**	
	$\mathcal{S} \leftarrow \mathsf{rng}(\widetilde{\Pi}[T, \cdot, \cdot])$	
	$C_1 \| C_2 \leftarrow\!\$\, \{0,1\}^{n+\lvert M\rvert} \setminus \mathcal{S}$	
	$\widetilde{\Pi}[T, Z, M] \leftarrow (C_1, C_2)$	
	$\widetilde{\Pi}^{-1}[T, C_1, C_2] \leftarrow (Z, M)$	
	$(C_1, C_2) \leftarrow \widetilde{\Pi}[T, Z, M]$	
	$\mathcal{C} \leftarrow \mathcal{C} \cup \{(N, H, C_1, C_2)\}$	
	return (C_1, C_2)	

Fig. 20: The games $\mathbf{G}_1$ and $\mathbf{G}_2$ for the proof of Theorem 2 (EtE scheme). $\mathbf{G}_2$ does not contain the boxed code.

$\mathbf{G}_2$: This is the ideal world of the MRAE game, and it is given in Fig. 20. We obtain the game $\mathbf{G}_2$ by removing the boxed code from the game $\mathbf{G}_1$. By removing the boxed code, the ciphertext pair (C_1, C_2) returned by the encryption oracle in the game $\mathbf{G}_2$ will always be sampled uniformly at random.

We bound $\mathcal{A}$'s distinguishing advantage by introducing an event bad and utilizing the fundamental lemma of game playing. The event bad is an event that a collision in the range of a lazily-sampled ideal cipher $\widetilde{\Pi}[T, \cdot, \cdot]$ occurs. It then holds that

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_1} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_2} \Rightarrow 1\right]\right| \leq \Pr[\mathsf{bad}] \leq \frac{q_e^2}{2^{n+m+1}}, \tag{9}$$

with the bound being a birthday bound on a variable-length domain of size at least $n + m$.

Finally, by combining the inequalities (8) and (9) one obtains the theorem bound. □

C.2 Proof of Theorem 3 (EtD Construction)

Proof. We prove the RUPAE security of the EtD construction by using a sequence of games. We will give the initialization and finalization code of the games only for the first two games for the sake of code readability. Without loss of generality, we assume the adversary $\mathcal{A}$ does not repeat queries to the Dec and Leak oracles.

$\mathbf{G}_0$: This is the real world of the RUPAE game, with an extra "bookkeeping" involving the set $\mathcal{Q}$. The game is given in Fig. 21.

$\mathbf{G}_1$: In the game $\mathbf{G}_1$, we replace the tweakable cipher $\widetilde{\mathsf{EE}}$ with an ideal cipher. We represent this ideal cipher with two independent but consistent tables $\widetilde{\Pi}$ and $\widetilde{\Pi}^{-1}$. The entries in tables $\widetilde{\Pi}$ and $\widetilde{\Pi}^{-1}$ will also have a value 'E','D' or 'L' as their first coordinate. We use these values to mark where the entry to the table was made ('E' means the entry was made during a query to the encryption oracle, 'D' during decryption, and 'L' during leakage, respectively). We additionally impose on the tables $\widetilde{\Pi}$ and $\widetilde{\Pi}^{-1}$ a requirement that if an entry $[\cdot, T, X, Y]$ is added to the table, any other entry that is indexed by (T, X, Y) is implicitly removed from the table. That is, it cannot happen that $\widetilde{\Pi}$ or $\widetilde{\Pi}^{-1}$ contains an entry $(\cdot, T, X, Y)$ for two different 'E','D' or 'L'.

We want to reduce the adversary's advantage in distinguishing the games $\mathbf{G}_0$ and $\mathbf{G}_1$ to the RPRP security of the underlying tweakable cipher. Let $\mathcal{B}$ be an adversary playing the RPRP game. It runs $\mathcal{A}$ and simulates the games $\mathbf{G}_0$ and $\mathbf{G}_1$ for it. Note that the adversary $\mathcal{A}$ can make $\mathcal{B}$ trigger a "forbidden" RPRP query. Namely, suppose that during the simulation of Enc oracle, the triple (N, H, M) encodes to Z which was either:

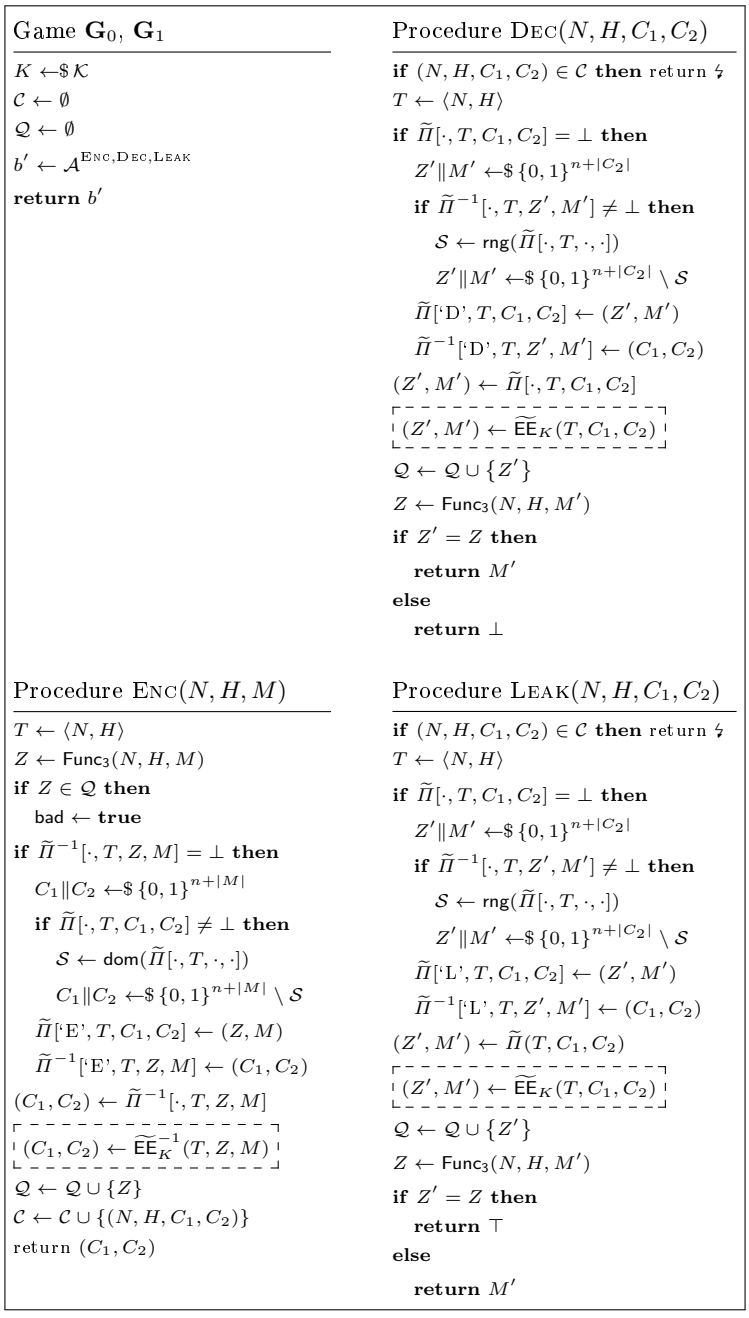

Fig. 21: The games $\mathbf{G}_0$ and $\mathbf{G}_1$ for the proof of Theorem 3 (EtD scheme). $\mathbf{G}_1$ does not contain the boxed code.

1. Already used as a left input to the decipher algorithm (in some previous encryption oracle query), or
2. Already output by the encipher algorithm (in some previous decryption or leakage oracle query).

Both of those would lead to a forbidden RPRP query. Therefore, we introduce the set $\mathcal{Q}$, to be able to explicitly mark when the "bad" event happens. Formally, we denote this bad event with E and define it as

$$E\text{: an event that adversary } \mathcal{A} \text{ makes an encryption query } (N,H,M) \text{ s.t. } \mathsf{Func}_3(N,H,M) \in \mathcal{Q}.$$

Let E_0 be an event that E occurs in the game $\mathbf{G}_0$, and let E_1 be an event that E occurs in the game $\mathbf{G}_1$. We distinguish here these two events since in $\mathbf{G}_0$ the real cipher is used for generating values stored in $\mathcal{Q}$. On the other hand, in $\mathbf{G}_1$ the ideal cipher is used. Next, we use the difference lemma given in Appendix A.3 to bound $\mathcal{A}$'s distinguishing advantage as follows.

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_0} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_1} \Rightarrow 1\right]\right| \leq \left|\Pr\left[\mathcal{A}^{\mathbf{G}_0} \Rightarrow 1 \wedge \neg E_0\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_1} \Rightarrow 1 \wedge \neg E_1\right]\right| + \max\{\Pr[E_0], \Pr[E_1]\}.$$

If the events E_0 and E_1 do not happen, the absolute value term reduces to the $\mathcal{B}$'s RPRP advantage. The adversary $\mathcal{B}$ works as follows. Adversary $\mathcal{B}$ is answering $\mathcal{A}$'s encryption queries by calculating T and Z itself, and then forwarding (T,Z,M) to its backward oracle De. The result (C_1,C_2) is forwarded back to $\mathcal{A}$. It simulates the decryption and leakage oracle for $\mathcal{A}$ as follows. Upon receiving a query (N,H,C_1,C_2), $\mathcal{B}$ calculates T and then queries its forward oracle En with (T,C_1,C_2). When it receives the result, it calculates Z, checks if it is equal to Z' and returns to $\mathcal{A}$ the appropriate answer. The adversary $\mathcal{B}$ guesses what $\mathcal{A}$ guesses. Assuming the event E does not occur, the adversary $\mathcal{B}$ simulates the games for $\mathcal{A}$ perfectly. Hence, it holds that

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_0} \Rightarrow 1 \wedge \neg E_0\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_1} \Rightarrow 1 \wedge \neg E_1\right]\right| \leq \mathbf{Adv}^{\mathrm{rprp}}_{\widetilde{\mathsf{EE}}}(\mathcal{B},1).$$

In order to bound the term $\max\{\Pr[E_0],\Pr[E_1]\}$, we construct another adversary $\mathcal{C}$ that is playing the RPRP game. The adversary $\mathcal{C}$ will internally keep track of the set $\mathcal{Q}$ during the simulation of $\mathbf{G}_0$ and $\mathbf{G}_1$ for the adversary $\mathcal{A}$. The decryption and leakage oracles are simulated the same way the adversary $\mathcal{B}$ simulates them. As for the encryption oracle and $\mathcal{A}$'s query (N,H,M), $\mathcal{C}$ first checks if $\mathsf{Func}_3(N,H,M)$ would be in $\mathcal{Q}$. If it would, the adversary $\mathcal{C}$ halts and outputs 1. Otherwise, $\mathcal{C}$ continues simulating the oracle. In the end, the adversary $\mathcal{C}$ outputs 0. Note that the adversary $\mathcal{C}$ needs to keep track of the set $\mathcal{Q}$ (and it indeed can) in order to be a valid RPRP adversary, that is, an adversary that does not make a forbidden query.

We see that $\mathcal{C}$ outputs 1 if it was interacting with the real cipher only when the event E_0 occurs. Similarly, it outputs 1 if it was interacting with the ideal cipher only when the event E_1 occurs. Therefore,

$$|\Pr[E_0] - \Pr[E_1]| \leq \mathbf{Adv}^{\mathrm{rprp}}_{\widetilde{\mathsf{EE}}}(\mathcal{C},1).$$

Furthermore, it holds that $\max\{\Pr[E_0],\Pr[E_1]\} \leq \mathbf{Adv}^{\mathrm{rprp}}_{\widetilde{\mathsf{EE}}}(\mathcal{C},1) + \Pr[E_1]$. Summarizing the findings so far, we have that

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_0} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_1} \Rightarrow 1\right]\right| \leq \mathbf{Adv}^{\mathrm{rprp}}_{\widetilde{\mathsf{EE}}}(\mathcal{B},1) + \mathbf{Adv}^{\mathrm{rprp}}_{\widetilde{\mathsf{EE}}}(\mathcal{C},1) + \Pr[E_1]. \tag{10}$$

We are left to bound the term $\Pr[E_1]$. We delay this calculation until we arrive at the game $\mathbf{G}_4$. Letting E_i correspond to the event of E occuring in the game $\mathbf{G}_i$, we first bound the term $|\Pr[E_1] - \Pr[E_4]|$. By a simple subtraction and addition of the same term $\Pr[E_i]$, for $i \in \{2,3\}$, it holds that

$$\begin{aligned}|\Pr[E_1] - \Pr[E_4]| &= |\Pr[E_1] - \Pr[E_2] + \Pr[E_2] - \Pr[E_3] + \Pr[E_3] - \Pr[E_4]| \\ &\leq |\Pr[E_1] - \Pr[E_2]| + |\Pr[E_2] - \Pr[E_3]| + |\Pr[E_3] - \Pr[E_4]|.\end{aligned}$$

Then, with the transfer of $\Pr[E_4]$ to the right-hand side of the equation we get

$$\Pr[E_1] \leq |\Pr[E_1] - \Pr[E_2]| + |\Pr[E_2] - \Pr[E_3]| + |\Pr[E_3] - \Pr[E_4]| + \Pr[E_4].$$

The absolute value terms can be bounded by the adversary's advantage in distinguishing the corresponding games, effectively doubling its advantage in distinguishing the games $\mathbf{G}_1$ and $\mathbf{G}_2$, $\mathbf{G}_2$ and $\mathbf{G}_3$, and $\mathbf{G}_3$ and $\mathbf{G}_4$. In the end, we are left to bound the term $\Pr[E_4]$ in the game $\mathbf{G}_4$.

$\mathbf{G}_2$: In the game $\mathbf{G}_2$, we apply Lemma 1 to make $\widetilde{\Pi}$ be a tweakable two-sided random function instead of an ideal cipher. In principle, we achieve that by removing the code that resamples values if they are in the domain or range of $\widetilde{\Pi}$ from $\mathbf{G}_1$. Let $\mathcal{D}$ be an adversary with access to either ideal cipher or tweakable two-sided random function, both having the tweak space $\mathcal{T}$ and over the domain $\mathcal{X}_L \times \mathcal{X}_R$. The adversary $\mathcal{D}$ can simulate the games $\mathbf{G}_1$ and $\mathbf{G}_2$ for $\mathcal{A}$. For the sake of proof simplicity, we briefly describe how $\mathcal{D}$ works and do not give the details. The adversary $\mathcal{D}$ will construct the tables $\widetilde{\Pi}$ and $\widetilde{\Pi}^{-1}$ in which it will store the results of its own queries to the forward or backward oracle that it made while simulating the oracles Enc, Dec, and Leak. In case the adversary $\mathcal{A}$ made a query such that the corresponding entry in the table $\widetilde{\Pi}$ or $\widetilde{\Pi}^{-1}$ already exists, $\mathcal{D}$ will not make a repeated query to its own oracles but just read the needed result from either table $\widetilde{\Pi}$ or $\widetilde{\Pi}^{-1}$. $\mathcal{D}$ will make at most q_{e} queries to its forward oracle and at most $q_{\mathrm{d}} + q_{\mathrm{l}}$ queries to its backward oracle. Applying the result of Lemma 1, we can reduce $\mathcal{A}$'s distinguishing advantage between the games $\mathbf{G}_1$ and $\mathbf{G}_2$ to $\mathcal{D}$'s distinguishing advantage between ideal cipher and tweakable two-sided random function. Therefore, it holds

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_1} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_2} \Rightarrow 1\right]\right| \leq \frac{(q_{\mathrm{e}} + q_{\mathrm{d}} + q_{\mathrm{l}})^2}{2^{n+m+1}}. \tag{11}$$

$\mathbf{G}_3$: In the game $\mathbf{G}_3$, we intend to remove the dependency in the decryption and verification oracles on the entries made in the encryption oracle. We start by transforming all the procedures in the game $\mathbf{G}_2$ to obtain the game $\mathbf{G}_2^*$. The games $\mathbf{G}_2^*$ and $\mathbf{G}_3$ are given in Fig. 22. We explain the transformation for the encryption oracle, and the same principle holds for the other two oracles. In $\mathbf{G}_2$ the encryption oracle first checks if the entry for the triple (T, Z, M) is defined in the table $\widetilde{\Pi}^{-1}$, if it is not, the oracle samples the ciphertext at random, and fills the corresponding entries in $\widetilde{\Pi}$ and $\widetilde{\Pi}^{-1}$. After exiting the **if** branch, it sets (C_1, C_2) to $\widetilde{\Pi}^{-1}[\cdot, T, Z, M]$ which also covers the case if the entry for the triple (T, Z, M) in $\widetilde{\Pi}^{-1}$ was already defined before entering Enc oracle. In $\mathbf{G}_2^*$, the encryption oracle first samples the ciphertext, and then overwrites it with $\widetilde{\Pi}^{-1}[\cdot, T, Z, M]$ if the entry (for the corresponding 'D', 'L' or 'E') already exists. If the entry does not exist, the encryption oracle fills the appropriate entries in $\widetilde{\Pi}$ and $\widetilde{\Pi}^{-1}$. The encryption oracles behave the same in both games and deliver the identical result for any queried triple (N, H, M). The analogous explanation holds for the decryption and leakage oracle. In addition, we add the flags $\mathsf{bad}_{L,2}$ and $\mathsf{bad}_{D,2}$ during this code transformation transformation. The games $\mathbf{G}_2$ and $\mathbf{G}_2^*$ are hence equivalent.

Going further, the games $\mathbf{G}_2^*$ and $\mathbf{G}_3$ are identical-until-bad games, where the game $\mathbf{G}_3$ does not contain the boxed code from $\mathbf{G}_2^*$. $\Pr[\mathsf{bad}_{D,2}] = 0$, since otherwise (N, H, C_1, C_2) that "triggers" $\widetilde{\Pi}[\text{'E'}, T, C_1, C_2]$ would be a forward query from Enc, which is forbidden. Same holds for $\Pr[\mathsf{bad}_{L,2}]$. Thus, we have

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_2} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_3} \Rightarrow 1\right]\right| = \left|\Pr\left[\mathcal{A}^{\mathbf{G}_2^*} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_3} \Rightarrow 1\right]\right| \leq \Pr[\mathsf{bad}_{D,2}] + \Pr[\mathsf{bad}_{L,2}] = 0. \tag{12}$$

$\mathbf{G}_4$: In the game $\mathbf{G}_4$, we want to remove the possibility of a forgery happening in either the decryption or leakage oracle. We add flags $\mathsf{bad}_{D,3}$ and $\mathsf{bad}_{L,3}$ to the game $\mathbf{G}_3$, and obtain an equivalent game $\mathbf{G}_3^*$. We then remove the boxed code from $\mathbf{G}_3^*$ to arrive at the game $\mathbf{G}_4$. The games $\mathbf{G}_3^*$ and $\mathbf{G}_4$ are given in Fig. 23. Using the identical-until-bad games approach, we bound the adversary's advantage in distinguishing between the games $\mathbf{G}_3^*$ and $\mathbf{G}_4$ with $\Pr[\mathsf{bad}_{D,3}] + \Pr[\mathsf{bad}_{L,3}]$. Let us denote with $\mathsf{bad}_{D,3}^i$ the event that the flag $\mathsf{bad}_{D,3}$ was set during i-th query to Dec oracle.

$$\Pr[\mathsf{bad}_{D,3}] \leq \sum_{i=1}^{q_{\mathrm{d}}} \mathsf{bad}_{D,3}^i = \sum_{i=1}^{q_{\mathrm{d}}} \frac{1}{2^n} = \frac{q_{\mathrm{d}}}{2^n},$$

since the probability that calculated Z is equal to a randomly generated Z' at the start of each Dec query is exactly $1/2^n$. Using the same principle, we know $\Pr[\mathsf{bad}_{L,3}] \leq q_{\mathrm{l}}/2^n$. In total,

$$\begin{aligned}\left|\Pr\left[\mathcal{A}^{\mathbf{G}_3} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_4} \Rightarrow 1\right]\right| &= \left|\Pr\left[\mathcal{A}^{\mathbf{G}_3^*} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_4} \Rightarrow 1\right]\right| \\ &\leq \Pr[\mathsf{bad}_{D,3}] + \Pr[\mathsf{bad}_{L,3}] \leq \frac{q_{\mathrm{d}} + q_{\mathrm{l}}}{2^n}.\end{aligned} \tag{13}$$

Finally, as promised, we bound the probability of the event E_4 happening. In order to do that, we first alter the process of adding elements to the set $\mathcal{Q}$ by adding a label 'R' or 'F' together with the element. The label 'R' is added during the expansion of $\mathcal{Q}$ in the encryption procedure, while the label 'F' is

$\mathbf{G}_2^*$, $\mathbf{G}_3$

Procedure $\textsc{Enc}(N, H, M)$

$T \leftarrow \langle N, H \rangle$
$Z \leftarrow \mathsf{Func}_3(N, H, M)$
if $Z \in \mathcal{Q}$ **then**
  $\mathsf{bad} \leftarrow$ **true**
$C_1 \| C_2 \leftarrow\$ \{0,1\}^{n+|M|}$
if $\widetilde{\Pi}^{-1}[\text{'D'}, T, Z, M] \neq \perp$ **then**
  $(C_1, C_2) \leftarrow \widetilde{\Pi}^{-1}[\text{'D'}, T, Z, M]$
if $\widetilde{\Pi}^{-1}[\text{'L'}, T, Z, M] \neq \perp$ **then**
  $(C_1, C_2) \leftarrow \widetilde{\Pi}^{-1}[\text{'L'}, T, Z, M]$
if $\widetilde{\Pi}^{-1}[\text{'E'}, T, Z, M] \neq \perp$ **then**
  $(C_1, C_2) \leftarrow \widetilde{\Pi}^{-1}[\text{'E'}, T, Z, M]$
if $\widetilde{\Pi}^{-1}[\cdot, T, Z, M] = \perp$ **then**
  $\widetilde{\Pi}[\text{'E'}, T, C_1, C_2] \leftarrow (Z, M)$
  $\widetilde{\Pi}^{-1}[\text{'E'}, T, Z, M] \leftarrow (C_1, C_2)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{Z\}$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(N, H, C_1, C_2)\}$
return (C_1, C_2)

Procedure $\textsc{Dec}(N, H, C_1, C_2)$

if $(N, H, C_1, C_2) \in \mathcal{C}$ **then**
  return $\lightning$
$T \leftarrow \langle N, H \rangle$
$Z' \| M' \leftarrow\$ \{0,1\}^{n+|C_2|}$
if $\widetilde{\Pi}[\text{'E'}, T, C_1, C_2] \neq \perp$ **then**
  $\mathsf{bad}_{D,2} \leftarrow$ **true**
  $(Z', M') \leftarrow \widetilde{\Pi}[\text{'E'}, T, C_1, C_2]$
if $\widetilde{\Pi}[\text{'L'}, T, C_1, C_2] \neq \perp$ **then**
  $(Z', M') \leftarrow \widetilde{\Pi}[\text{'L'}, T, C_1, C_2]$
if $\widetilde{\Pi}[\text{'D'}, T, C_1, C_2] \neq \perp$ **then**
  $(Z', M') \leftarrow \widetilde{\Pi}[\text{'D'}, T, C_1, C_2]$
if $\widetilde{\Pi}[\cdot, T, C_1, C_2] = \perp$ **then**
  $\widetilde{\Pi}[\text{'D'}, T, C_1, C_2] \leftarrow (Z', M')$
  $\widetilde{\Pi}^{-1}[\text{'D'}, T, Z', M'] \leftarrow (C_1, C_2)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{Z'\}$
$Z \leftarrow \mathsf{Func}_3(N, H, M')$
if $Z' = Z$ **then**
  return M'
else
  return $\perp$

Procedure $\textsc{Leak}(N, H, C_1, C_2)$

if $(N, H, C_1, C_2) \in \mathcal{C}$ **then**
  return $\lightning$
$T \leftarrow \langle N, H \rangle$
$Z' \| M' \leftarrow\$ \{0,1\}^{n+|C_2|}$
if $\widetilde{\Pi}[\text{'E'}, T, C_1, C_2] \neq \perp$ **then**
  $\mathsf{bad}_{L,2} \leftarrow$ **true**
  $(Z', M') \leftarrow \widetilde{\Pi}[\text{'E'}, T, C_1, C_2]$
if $\widetilde{\Pi}[\text{'D'}, T, C_1, C_2] \neq \perp$ **then**
  $(Z', M') \leftarrow \widetilde{\Pi}[\text{'D'}, T, C_1, C_2]$
if $\widetilde{\Pi}[\text{'L'}, T, C_1, C_2] \neq \perp$ **then**
  $(Z', M') \leftarrow \widetilde{\Pi}[\text{'L'}, T, C_1, C_2]$
if $\widetilde{\Pi}[\cdot, T, C_1, C_2] = \perp$ **then**
  $\widetilde{\Pi}[\text{'L'}, T, C_1, C_2] \leftarrow (Z', M')$
  $\widetilde{\Pi}^{-1}[\text{'L'}, T, Z', M'] \leftarrow (C_1, C_2)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{Z'\}$
$Z \leftarrow \mathsf{Func}_3(N, H, M')$
if $Z' = Z$ **then**
  return $\top$
else
  return M'

Fig. 22: The games $\mathbf{G}_2^*$ and $\mathbf{G}_3$ for the proof of Theorem 3 (`EtD` scheme). $\mathbf{G}_3$ does not contain the boxed code.

added during the expansion of $\mathcal{Q}$ in the decryption and leakage procedures. The label should help us differentiate whether Z that would potentially trigger the flag bad is a repeat to the decipher algorithm left input (labeled with 'R') or is a forward from encipher algorithm output to the decipher algorithm left input (labeled with 'F'). Consequently, we are able to split the set $\mathcal{Q}$ into two sets $\mathcal{Q}_{\mathrm{R}}$ and $\mathcal{Q}_{\mathrm{F}}$, each of which contains elements with label 'R' and 'F', respectively.

Let R_4 be the event where, during some encryption oracle query, the flag bad is triggered by Z being in $\mathcal{Q}_{\mathrm{R}}$. Similarly, let F_4 be the event where, during some encryption oracle query, the flag bad is triggered by Z being in $\mathcal{Q}_{\mathrm{F}}$. We can than look at the event E_4 as being the union of R_4 and F_4. By the union bound, it holds that

$$\Pr[E_4] \leq \Pr[R_4] + \Pr[F_4].$$

We bound the probability of R_4 happening by constructing an adversary $\mathcal{E}_{\mathsf{col}}$ that will play the collision experiment for the function Func_3. The adversary $\mathcal{E}_{\mathsf{col}}$ internally runs $\mathcal{A}$ and simulates the game $\mathbf{G}_4$ for it while keeping a list of all the queries $\mathcal{A}$ makes. During the simulation of the encryption procedure, $\mathcal{E}_{\mathsf{col}}$ looks at whether for a new query (N, H, M) a collision in the locally simulated set $\mathcal{Q}_{\mathrm{R}}$ would occur. If yes, the adversary $\mathcal{E}_{\mathsf{col}}$ aborts and outputs the corresponding colliding triples. Since the adversary $\mathcal{A}$ is nonce-respecting, such collision would also be a winning pair for the collision experiment. Clearly, the $\mathcal{E}_{\mathsf{col}}$'s probability of finding a collision equals the probability of $\mathcal{A}$ making a valid encryption query (N, H, M) such that $\mathsf{Func}_3(N, H, M)$ is already in the set $\mathcal{Q}_{\mathrm{R}}$. The former (and thus the latter) is bounded by δ.

$$\Pr[R_4] = \Pr[\mathcal{E}_{\mathsf{col}} \text{ outputs a valid collision}] \leq \delta.$$

Going further, we define an event F_4^i as the event that F_4 occurs within $\mathcal{A}$'s first i encryption queries. Then we can write event F_4 as

$$F_4 = F_4^1 \vee (F_4^2 \wedge \neg F_4^1) \vee \cdots \vee (F_4^{q_{\mathrm{e}}} \wedge \neg F_4^{q_{\mathrm{e}}-1}).$$

By the union bound and the definition of conditional probability, it holds

$$\Pr[F_4] \leq \sum_{i=1}^{q_{\mathrm{e}}} \Pr\left[F_4^i \wedge \neg F_4^{i-1}\right] = \sum_{i=1}^{q_{\mathrm{e}}} \Pr\left[F_4^i \,\middle|\, \neg F_4^{i-1}\right] \Pr\left[\neg F_4^{i-1}\right] \leq \sum_{i=1}^{q_{\mathrm{e}}} \Pr\left[F_4^i \,\middle|\, \neg F_4^{i-1}\right].$$

The event in the final summand above corresponds to an event that for $\mathcal{A}$'s i-th encryption query (N_i, H_i, M_i), it happens that $\mathsf{Func}_3(N_i, H_i, M_i) \in \mathcal{Q}_{\mathrm{F}}$. Therefore, we expand the single summand $\Pr\left[F_4^i \,\middle|\, \neg F_4^{i-1}\right]$ over all elements $Z'_j \in \mathcal{Q}_{\mathrm{F}}$ and get

$$\Pr[F_4] \leq \sum_{i=1}^{q_{\mathrm{e}}} \sum_{Z'_j \in \mathcal{Q}_{\mathrm{F}}} \Pr\left[\mathsf{Func}_3(N_i, H_i, M_i) = Z'_j \,\middle|\, \neg F_4^{i-1}\right].$$

To bound the term

$$\sum_{Z'_j \in \mathcal{Q}_{\mathrm{F}}} \Pr\left[\mathsf{Func}_3(N_i, H_i, M_i) = Z'_j \,\middle|\, \neg F_4^{i-1}\right]$$

we use the principle of deferred decision. We "fix" the value of $\mathsf{Func}_3(N_i, H_i, M_i)$ and think of the values Z'_j as being randomly sampled at the moment of check if $\mathsf{Func}_3(N_i, H_i, M_i) \in \mathcal{Q}_{\mathrm{F}}$. Conditioned on the event F_4^{i-1} not happening, $i-1$ random values are excluded from the sample space of random variables Z'_j, since in that case one knows that for none of the previous encryption queries, i.e. (N_k, H_k, M_k), $k < i$, the value $\mathsf{Func}_3(N_k, H_k, M_k)$ was contained in the set $\mathcal{Q}_{\mathrm{F}}$. Therefore, the probability that $\mathsf{Func}_3(N_i, H_i, M_i) \in \mathcal{Q}_{\mathrm{F}}$, given the event F_4^{i-1} did not occur, is bounded by

$$\frac{q_{\mathrm{d}} + q_{\mathrm{l}}}{2^n - (i-1)},$$

as the size of the set $\mathcal{Q}_{\mathrm{F}}$ at any point in the game is at most $q_{\mathrm{d}} + q_{\mathrm{l}}$. That leads us to the final bound for the event F_4 occurring,

$$\Pr[F_4] \leq \sum_{i=1}^{q_{\mathrm{e}}} \frac{q_{\mathrm{d}} + q_{\mathrm{l}}}{2^n - (i-1)} \leq \sum_{i=1}^{q_{\mathrm{e}}} \frac{q_{\mathrm{d}} + q_{\mathrm{l}}}{2^n - q_{\mathrm{e}}} = \frac{q_{\mathrm{e}}(q_{\mathrm{d}} + q_{\mathrm{l}})}{2^n - q_{\mathrm{e}}} \leq \frac{q_{\mathrm{e}}(q_{\mathrm{d}} + q_{\mathrm{l}})}{2^{n-1}},$$

under the assumption $q_{\mathrm{e}} \leq 2^{n-1}$.

$\mathbf{G}_3^*$, $\mathbf{G}_4$

Procedure $\textsc{Enc}(N, H, M)$

$T \leftarrow \langle N, H \rangle$
$Z \leftarrow \mathsf{Func}_3(N, H, M)$
if $(Z, \cdot) \in \mathcal{Q}$ **then**
  $\mathsf{bad} \leftarrow$ **true**
$C_1 \| C_2 \leftarrow\$ \{0,1\}^{n+|M|}$
if $\widetilde{\Pi}^{-1}[\text{`D'}, T, Z, M] \neq \perp$ **then**
  $(C_1, C_2) \leftarrow \widetilde{\Pi}^{-1}[\text{`D'}, T, Z, M]$
if $\widetilde{\Pi}^{-1}[\text{`L'}, T, Z, M] \neq \perp$ **then**
  $(C_1, C_2) \leftarrow \widetilde{\Pi}^{-1}[\text{`L'}, T, Z, M]$
if $\widetilde{\Pi}^{-1}[\text{`E'}, T, Z, M] \neq \perp$ **then**
  $(C_1, C_2) \leftarrow \widetilde{\Pi}^{-1}[\text{`E'}, T, Z, M]$
if $\widetilde{\Pi}^{-1}[\cdot, T, Z, M] = \perp$ **then**
  $\widetilde{\Pi}[\text{`E'}, T, C_1, C_2] \leftarrow (Z, M)$
  $\widetilde{\Pi}^{-1}[\text{`E'}, T, Z, M] \leftarrow (C_1, C_2)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(Z, \text{`R'})\}$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(N, H, C_1, C_2)\}$
return (C_1, C_2)

Procedure $\textsc{Dec}(N, H, C_1, C_2)$

if $(N, H, C_1, C_2) \in \mathcal{C}$ **then**
  return $\lightning$
$T \leftarrow \langle N, H \rangle$
$Z' \| M' \leftarrow\$ \{0,1\}^{n+|C_2|}$
if $\widetilde{\Pi}[\text{`L'}, T, C_1, C_2] \neq \perp$ **then**
  $(Z', M') \leftarrow \widetilde{\Pi}[\text{`L'}, T, C_1, C_2]$
if $\widetilde{\Pi}[\text{`D'}, T, C_1, C_2] \neq \perp$ **then**
  $(Z', M') \leftarrow \widetilde{\Pi}[\text{`D'}, T, C_1, C_2]$
if $\widetilde{\Pi}[\cdot, T, C_1, C_2] = \perp$ **then**
  $\widetilde{\Pi}[\text{`D'}, T, C_1, C_2] \leftarrow (Z', M')$
  $\widetilde{\Pi}^{-1}[\text{`D'}, T, Z', M'] \leftarrow (C_1, C_2)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(Z', \text{`F'})\}$
$Z \leftarrow \mathsf{Func}_3(N, H, M')$
if $Z' = Z$ **then**
  $\mathsf{bad}_{D,3} \leftarrow$ **true**
  return M'
else
  return $\perp$

Procedure $\textsc{Leak}(N, H, C_1, C_2)$

if $(N, H, C_1, C_2) \in \mathcal{C}$ **then**
  return $\lightning$
$T \leftarrow \langle N, H \rangle$
$Z' \| M' \leftarrow\$ \{0,1\}^{n+|C_2|}$
if $\widetilde{\Pi}[\text{`D'}, T, C_1, C_2] \neq \perp$ **then**
  $(Z', M') \leftarrow \widetilde{\Pi}[\text{`D'}, T, C_1, C_2]$
if $\widetilde{\Pi}[\text{`L'}, T, C_1, C_2] \neq \perp$ **then**
  $(Z', M') \leftarrow \widetilde{\Pi}[\text{`L'}, T, C_1, C_2]$
if $\widetilde{\Pi}[\cdot, T, C_1, C_2] = \perp$ **then**
  $\widetilde{\Pi}[\text{`L'}, T, C_1, C_2] \leftarrow (Z', M')$
  $\widetilde{\Pi}^{-1}[\text{`L'}, T, Z', M'] \leftarrow (C_1, C_2)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(Z', \text{`F'})\}$
$Z \leftarrow \mathsf{Func}_3(N, H, M')$
if $Z' = Z$ **then**
  $\mathsf{bad}_{L,3} \leftarrow$ **true**
  return $\top$
else
  return M'

Fig. 23: The games $\mathbf{G}_3^*$ and $\mathbf{G}_4$ for the proof of Theorem 3 (EtD scheme). $\mathbf{G}_4$ does not contain the boxed code.

$\mathbf{G}_4^*, \mathbf{G}_5$

Procedure $\textsc{Enc}(N, H, M)$

$T \leftarrow \langle N, H \rangle$
$Z \leftarrow \mathsf{Func}_3(N, H, M)$
$C_1 \| C_2 \leftarrow\$ \{0,1\}^{n+|M|}$
if $\widetilde{\Pi}^{-1}[\text{'D'}, T, Z, M] \neq \bot$ **then**
  $\mathsf{bad}_{E,2} \leftarrow$ **true**
  $(C_1, C_2) \leftarrow \widetilde{\Pi}^{-1}[\text{'D'}, T, Z, M]$
if $\widetilde{\Pi}^{-1}[\text{'L'}, T, Z, M] \neq \bot$ **then**
  $\mathsf{bad}_{E,3} \leftarrow$ **true**
  $(C_1, C_2) \leftarrow \widetilde{\Pi}^{-1}[\text{'L'}, T, Z, M]$
if $\widetilde{\Pi}^{-1}[\text{'E'}, T, Z, M] \neq \bot$ **then**
  $(C_1, C_2) \leftarrow \widetilde{\Pi}^{-1}[\text{'E'}, T, Z, M]$
if $\widetilde{\Pi}^{-1}[\cdot, T, Z, M] = \bot$ **then**
  $\widetilde{\Pi}[\text{'E'}, T, C_1, C_2] \leftarrow (Z, M)$
  $\widetilde{\Pi}^{-1}[\text{'E'}, T, Z, M] \leftarrow (C_1, C_2)$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(N, H, C_1, C_2)\}$
return (C_1, C_2)

Procedure $\textsc{Dec}(N, H, C_1, C_2)$

if $(N, H, C_1, C_2) \in \mathcal{C}$ **then**
  return $\lightning$
$T \leftarrow \langle N, H \rangle$
$Z' \| M' \leftarrow\$ \{0,1\}^{n+|C_2|}$
if $\widetilde{\Pi}[\text{'L'}, T, C_1, C_2] \neq \bot$ **then**
  $(Z', M') \leftarrow \widetilde{\Pi}[\text{'L'}, T, C_1, C_2]$
if $\widetilde{\Pi}[\text{'D'}, T, C_1, C_2] \neq \bot$ **then**
  $(Z', M') \leftarrow \widetilde{\Pi}[\text{'D'}, T, C_1, C_2]$
if $\widetilde{\Pi}[\cdot, T, C_1, C_2] = \bot$ **then**
  $\widetilde{\Pi}[\text{'D'}, T, C_1, C_2] \leftarrow (Z', M')$
  $\widetilde{\Pi}^{-1}[\text{'D'}, T, Z', M'] \leftarrow (C_1, C_2)$
return $\bot$

Procedure $\textsc{Leak}(N, H, C_1, C_2)$

if $(N, H, C_1, C_2) \in \mathcal{C}$ **then**
  return $\lightning$
$T \leftarrow \langle N, H \rangle$
$Z' \| M' \leftarrow\$ \{0,1\}^{n+|C_2|}$
if $\widetilde{\Pi}[\text{'D'}, T, C_1, C_2] \neq \bot$ **then**
  $(Z', M') \leftarrow \widetilde{\Pi}[\text{'D'}, T, C_1, C_2]$
if $\widetilde{\Pi}[\text{'L'}, T, C_1, C_2] \neq \bot$ **then**
  $(Z', M') \leftarrow \widetilde{\Pi}[\text{'L'}, T, C_1, C_2]$
if $\widetilde{\Pi}[\cdot, T, C_1, C_2] = \bot$ **then**
  $\widetilde{\Pi}[\text{'L'}, T, C_1, C_2] \leftarrow (Z', M')$
  $\widetilde{\Pi}^{-1}[\text{'L'}, T, Z', M'] \leftarrow (C_1, C_2)$
return M'

Fig. 24: The games $\mathbf{G}_4^*$ and $\mathbf{G}_5$ for the proof of Theorem 3 (EtD scheme). $\mathbf{G}_5$ does not contain the boxed code.

Finally, we can bound the probability of E_4 happening. It holds that

$$\Pr[E_4] \leq \Pr[R_4] + \Pr[F_4] \leq \delta + \frac{q_e(q_d + q_l)}{2^{n-1}}.$$

Together with the bounds in (11), (12) and (13), we arrive at the final bound for $\Pr[E_1]$,

$$\Pr[E_1] \leq \frac{(q_e + q_d + q_l)^2}{2^{n+m+1}} + \frac{q_d + q_l}{2^n} + \delta + \frac{q_e(q_d + q_l)}{2^{n-1}}. \tag{14}$$

$\mathbf{G}_5$: In the game $\mathbf{G}_5$, we intend to remove the dependency in the encryption oracle on the entries made in the decryption and verification oracles. Towards this goal, we add the flags $\mathsf{bad}_{E,2}$ and $\mathsf{bad}_{E,3}$ to the game $\mathbf{G}_4$ to obtain an equivalent game $\mathbf{G}_4^*$. In the hop to $\mathbf{G}_5$, we remove the boxed code from $\mathbf{G}_4^*$. The games $\mathbf{G}_4^*$ and $\mathbf{G}_5$ are given in Fig. 24. Using the identical-until-bad games paradigm, we bound the adversary's advantage in distinguishing the games $\mathbf{G}_4^*$ and $\mathbf{G}_5$ with $\Pr[\mathsf{bad}_{E,2}] + \Pr[\mathsf{bad}_{E,3}]$.

Suppose $\widetilde{\Pi}^{-1}[\text{'D'}, T, Z', M']$ was set in some query to the decryption oracle. The pair (N, H) that the adversary submits to the decryption oracle in that query uniquely determines T. If $\mathcal{A}$ aims to "trigger" the flag $\mathsf{bad}_{E,2}$, it needs to ask (N, H, M) to the encryption oracle, for some M from the message space. Moreover, the left part Z would have to be equal to the randomly generated Z' in that previous query to

the decryption oracle. In addition, the adversary may try to increase its chances of guessing by making multiple decryption queries with the same nonce and the header, which would lead to multiple defined values $\widetilde{\Pi}^{-1}[\text{'D'}, T, \cdot, \cdot]$. Since only unique nonces are queried to the encryption oracle, for all defined entries $\widetilde{\Pi}^{-1}[\text{'D'}, T, \cdot, \cdot]$ the adversary $\mathcal{A}$ has a single attempt to "guess" one of them. Therefore, it follows that

$$\Pr[\mathsf{bad}_{E,2}] \leq \sum_{i=1}^{q_\mathrm{e}} \frac{q_\mathrm{d}}{2^n} = \frac{q_\mathrm{e} q_\mathrm{d}}{2^n},$$

where $1/2^n$ is the probability of Z being equal to the randomly generated[4] Z' in a previous query to Dec, and the q_d in the numerator is the bound on the maximal number of Z', for which $\widetilde{\Pi}^{-1}[\text{'D'}, T, Z', \cdot]$ was defined. Similarly, it holds that $\Pr[\mathsf{bad}_{E,3}] \leq \frac{q_\mathrm{e} q_\mathrm{l}}{2^n}$. In total,

$$\begin{aligned} \left|\Pr\left[\mathcal{A}^{\mathbf{G}_4} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_5} \Rightarrow 1\right]\right| &= \left|\Pr\left[\mathcal{A}^{\mathbf{G}_4^*} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_5} \Rightarrow 1\right]\right| \\ &\leq \Pr[\mathsf{bad}_{E,2}] + \Pr[\mathsf{bad}_{E,3}] \leq \frac{q_\mathrm{e}(q_\mathrm{d} + q_\mathrm{l})}{2^n}. \end{aligned} \tag{15}$$

$\mathbf{G}_6$: This is the ideal world of the RUPAE game. We make a couple of changes in the game $\mathbf{G}_5$ to obtain $\mathbf{G}_6$ and then argue the games are equivalent. The game $\mathbf{G}_6$ is given in Fig. 25.

$\mathbf{G}_6$

Procedure Enc(N, H, M)	Procedure Dec(N, H, C_1, C_2)	Procedure Leak(N, H, C_1, C_2)
$T \leftarrow \langle N, H \rangle$	**if** $(N, H, C_1, C_2) \in \mathcal{C}$ **then**	**if** $(N, H, C_1, C_2) \in \mathcal{C}$ **then**
$Z \leftarrow \mathsf{Func}_3(N, H, M)$	return $\lightning$	return $\lightning$
$(C_1, C_2) \leftarrow \{0,1\}^{n+\lvert M\rvert}$	return $\perp$	$Z' \Vert M' \leftarrow\$ \{0,1\}^{n+\lvert C_2\rvert}$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(N, H, C_1, C_2)\}$		return M'
return (C_1, C_2)		

Fig. 25: The game $\mathbf{G}_6$ for the proof of Theorem 3 (EtD scheme).

The encryption oracle is reformatted such that it always returns random bits. In the game $\mathbf{G}_5$, the encryption oracle is not reading from the table entries made in Dec and Leak. It is only relying on the entries made during Enc. Moreover, the Dec and Leak oracles are not accessing entries in $\widetilde{\Pi}$ or $\widetilde{\Pi}^{-1}$ made in the encryption oracle. Thus, we can also remove the writing of the sampled ciphertext (C_1, C_2) to the $\widetilde{\Pi}$ and $\widetilde{\Pi}^{-1}$ tables. Note that Enc does not even need to check if $\widetilde{\Pi}^{-1}[\text{'E'}, T, Z, M]$ has been defined, as the assumption is the adversary always submits fresh nonce to the encryption oracle, and by the injective property of $\langle \cdot, \cdot \rangle$ and Func_3 it leads to a fresh triple (T, Z, M). Hence, it will never happen that $\widetilde{\Pi}^{-1}[\text{'E'}, T, Z, M]$ was already defined. When the redundant code is removed, the encryption oracle simply returns random bits.

The leakage oracle needs to be an independent simulator, so in the game hop from $\mathbf{G}_5$ to $\mathbf{G}_6$, we need to remove the reading from table $\widetilde{\Pi}[\text{'D'}, \cdot, \cdot, \cdot]$ in Leak oracle. In the case $\widetilde{\Pi}[\text{'D'}, T, C_1, C_2]$ is already defined, Leak needs to return M' that was sampled and written to the table in the decryption oracle. Anyhow, the M' sampled in Dec is a random bit string, not known to the adversary, since the decryption oracle in the game $\mathbf{G}_5$ does not leak any information. If we remove that part of code from Leak and rewrite the Leak oracle as just always returning random bits, the adversary's view does not change, as it will always get randomly sampled M', both in $\mathbf{G}_5$ and $\mathbf{G}_6$. The leakage oracle also does not need to read from $\widetilde{\Pi}[\text{'L'}, \cdot, \cdot, \cdot]$, since it does not make repeat queries and thus is sure that $\widetilde{\Pi}[\text{'L'}, T, C_1, C_2]$ has not been defined so far. As the decryption oracle also does not need to read from $\widetilde{\Pi}[\text{'L'}, \cdot, \cdot, \cdot]$ (see next paragraph), we can remove the writing to $\widetilde{\Pi}[\text{'L'}, \cdot, \cdot, \cdot]$. So we come to the final form of leakage oracle code, now an independent leakage simulator S.

In the decryption oracle, we can remove the reading from $\widetilde{\Pi}[\text{'L'}, \cdot, \cdot, \cdot]$, as the adversary always returns $\perp$, and reading from $\widetilde{\Pi}[\text{'L'}, \cdot, \cdot, \cdot]$ does not influence oracle's answer. There is also no need to either read from $\widetilde{\Pi}[\text{'D'}, \cdot, \cdot, \cdot]$ or to write to it.

[4] We again use the principle of deferred decision, and think of value Z' being generated after the value Z has been "fixed".

We showed the game $\mathbf{G}_5$ is equivalent to the game $\mathbf{G}_6$, the ideal world of the RUPAE game. It holds that

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_5} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_6} \Rightarrow 1\right]\right| = 0. \tag{16}$$

Combining the inequalities (10), (11), (12), (13), (14), (15) and (16), the theorem bound is achieved. □

D Appendix Section 5

D.1 Nonce-Set AEAD From Nonce-Hiding AEAD

$\mathsf{Enc}_K(N, H, M)$	$\mathsf{Dec}_K(\boldsymbol{W}, H, C)$
$C \leftarrow \overline{\mathsf{Enc}}_K(N, H, M)$	$(N, M) \leftarrow \overline{\mathsf{Dec}}_K(H, C)$
return C	**if** $(N, M) = \perp$ **then**
	return $(\perp, \perp)$
	else
	if $N \in \boldsymbol{W}$ **then**
	return (N, M)
	else
	return $(\perp, \perp)$

Fig. 26: Nonce-set AEAD from a nonce-hiding AEAD scheme $(\overline{\mathsf{Enc}}, \overline{\mathsf{Dec}})$.

D.2 Proof of Theorem 4 (AwN Construction)

Proof. We prove the MRAE security of the AwN construction by using the following sequence of games.

$\mathbf{G}_0$: This is the real world of the MRAE game and it is given in Fig. 27.

Game $\mathbf{G}_0$	Procedure $\textsc{Enc}(N, H, M)$
$K \leftarrow\$ \mathcal{K}$	$(C_1, C_2) \leftarrow \widetilde{\mathsf{EE}}_K(H, N \| \lfloor M \rfloor_{n-t}, \lceil M \rceil_{l-n+t})$
$\mathcal{C} \leftarrow \emptyset$	$\mathcal{C} \leftarrow \mathcal{C} \cup \{(N, H, C_1, C_2)\}$
$b' \leftarrow \mathcal{A}^{\textsc{Enc}, \textsc{Ver}}$	return (C_1, C_2)
return b'	
	Procedure $\textsc{Ver}(\boldsymbol{W}, H, C_1, C_2)$
	if $\mathcal{C} \cap \{(N, H, C_1, C_2) \mid N \in \boldsymbol{W}\} \neq \emptyset$ **then**
	return $\lightning$
	$(X_L, X_R) \leftarrow \widetilde{\mathsf{EE}}_K^{-1}(H, C_1, C_2)$
	$N' \leftarrow \lfloor X_L \rfloor_t$
	$v \leftarrow \perp$
	if $N' \in \boldsymbol{W}$ **then**
	$v \leftarrow \top$
	return v

Fig. 27: The game $\mathbf{G}_0$ for the proof of Theorem 4 (AwN scheme). l is the length of the message M.

$\mathbf{G}_1$: In the game $\mathbf{G}_1$, we replace the tweakable cipher $\widetilde{\mathsf{EE}}$ with a lazily-sampled ideal cipher $\widetilde{\Pi}$ in the encryption oracle and change the VER oracle always to return $\perp$. The game $\mathbf{G}_1$ is given in Fig. 28.

For any adversary $\mathcal{A}$ distinguishing between the games $\mathbf{G}_0$ and $\mathbf{G}_1$ we construct an RPRP adversary $\mathcal{B}$, that runs $\mathcal{A}$ and simulates $\mathbf{G}_0$ and $\mathbf{G}_1$ for $\mathcal{A}$. The adversary $\mathcal{B}$ works as follows. On $\mathcal{A}$'s encryption oracle query (N, H, M), the adversary $\mathcal{B}$ asks its enciphering oracle with the corresponding triple and returns the result back to $\mathcal{B}$, while also updating the set $\mathcal{C}$. On $\mathcal{A}$'s verification oracle query $(\boldsymbol{W}, H, C_1, C_2)$, $\mathcal{B}$ first tests, for all $N \in \boldsymbol{W}$, whether a tuple (N, H, C_1, C_2) is in $\mathcal{C}$. If it is, $\mathcal{B}$ returns $\lightning$. Otherwise, it goes on to construct a set

$$\boldsymbol{W}' = \left\{X \| Y \mid X \in \boldsymbol{W}, Y \in \{0,1\}^{n-t}\right\}.$$

Finally, the adversary $\mathcal{B}$ makes a guess oracle query $(H, C_1, C_2, \boldsymbol{W}')$. If GU oracle returns **true** $\mathcal{B}$ returns $\top$ back to $\mathcal{A}$, else $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$. Observe that the size of the set $\boldsymbol{W}'$ the adversary $\mathcal{B}$ passes to its GU oracle will be at most $w2^{n-t}$.

If $\mathcal{B}$ interacts with the real world of the RPRP game, it simulates the game $\mathbf{G}_0$ to $\mathcal{A}$. Otherwise, it simulates the game $\mathbf{G}_1$. We note that the verification oracle behaviour in the game $\mathbf{G}_0$ corresponds to the guess oracle code in the real world of RPRP game. It follows that

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_0} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_1} \Rightarrow 1\right]\right| \leq \mathbf{Adv}^{\mathrm{rprp}}_{\widetilde{\mathsf{EE}}}(\mathcal{B}, v). \tag{17}$$

$\mathbf{G}_2$: This is the ideal world of the MRAE game, and it is given in Fig. 28. We obtain the game $\mathbf{G}_2$ by removing the boxed code from the game $\mathbf{G}_1$. By removing the boxed code, the ciphertext pair (C_1, C_2) returned by the encryption oracle in the game $\mathbf{G}_2$ will always be sampled uniformly at random.

We bound $\mathcal{A}$'s distinguishing advantage by introducing an event bad and utilizing the fundamental lemma of game playing. The event bad is an event that a collision in the range of a lazily-sampled ideal cipher $\widetilde{\Pi}[T, \cdot, \cdot]$ occurs. It then holds that

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_1} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_2} \Rightarrow 1\right]\right| \leq \Pr[\mathsf{bad}] \leq \frac{q_{\mathrm{e}}^2}{2^{n+m+1}}, \tag{18}$$

with the bound being a birthday bound on a variable-length domain of size at least $n + m$. Finally, by combining the inequalities (17) and (18) one obtains the theorem bound. □

Game $\mathbf{G}_1$, $\mathbf{G}_2$

$K \leftarrow\$ \mathcal{K}$
$\mathcal{C} \leftarrow \emptyset$
$b' \leftarrow \mathcal{A}^{\textsc{Enc}, \textsc{Ver}}$
return b'

Procedure $\textsc{Enc}(N, H, M)$

if $\widetilde{\Pi}[H, N\|\lfloor M\rfloor_{n-t}, \lceil M\rceil_{l-n+t}] = \perp$ **then**
  $C_1\|C_2 \leftarrow\$ \{0,1\}^{t+|M|}$
  if $\widetilde{\Pi}^{-1}[H, C_1, C_2] \neq \perp$ **then**
    bad $\leftarrow$ **true**
    $\mathcal{S} \leftarrow \mathsf{rng}(\widetilde{\Pi}[H, \cdot, \cdot])$
    $C_1\|C_2 \leftarrow\$ \{0,1\}^{t+|M|} \setminus \mathcal{S}$
  $\widetilde{\Pi}[H, N\|\lfloor M\rfloor_{n-t}, \lceil M\rceil_{l-n+t}] \leftarrow (C_1, C_2)$
  $\widetilde{\Pi}^{-1}[H, C_1, C_2] \leftarrow (N\|\lfloor M\rfloor_{n-t}, \lceil M\rceil_{l-n+t})$
$(C_1, C_2) \leftarrow \widetilde{\Pi}[H, N\|\lfloor M\rfloor_{n-t}, \lceil M\rceil_{l-n+t}]$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(N, H, C_1, C_2)\}$
return (C_1, C_2)

Procedure $\textsc{Ver}(\boldsymbol{W}, H, C_1, C_2)$

if $\mathcal{C} \cap \{(N, H, C_1, C_2) \mid N \in \boldsymbol{W}\} \neq \emptyset$ **then**
  return $\lightning$
return $\perp$

Fig. 28: The games $\mathbf{G}_1$ and $\mathbf{G}_2$ for the proof of Theorem 4 (AwN scheme). $\mathbf{G}_2$ does not contain the boxed code. l is the length of the message M.

E Appendix Section 6

E.1 Robustness Game ROB

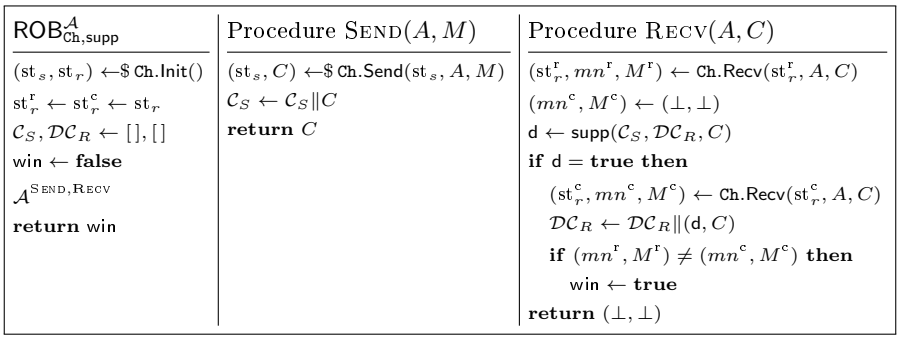

Fig. 29: The game ROB used to define robustness for channels.

E.2 Proof of Theorem 5 (Nonce-set Channel Construction Correctness)

Proof. We start by assuming the contrapositive is true. If the channel $\mathtt{Ch}_{\mathrm{NS}}$ is not correct, then either the underlying nonce-set AEAD NSE is not correct, or the underlying nonce-set processing scheme does not faithfully reproduce the support predicate supp. Suppose the adversary $\mathcal{A}$ successfully breaks the correctness of the channel, and without loss of generality, assume $\mathcal{A}$ queries $\textsc{Recv}$ oracle only with indices j that correspond to the supported ciphertexts. We then show the contrapositive by introducing two "complementary" reductions. The adversary $\mathcal{B}$, being a reduction to the faithfulness of NSP, will run $\mathcal{A}$ and simulate the correctness game to it. In this case, we show that under the assumption NSE is correct, the adversary $\mathcal{B}$ breaks faithfulness of NSP when $\mathcal{A}$ wins in the channel correctness game. The other case is the reduction to the correctness of NSE. In this case, we argue that if NSP faithfully reproduces the support predicate supp, and that the adversary $\mathcal{A}$ wins in the channel correctness game, the nonce-set scheme NSE would not be correct.

Reduction to the faithfulness of NSP*:* The adversary $\mathcal{B}$ needs to simulate CORR game (Fig. 11) to $\mathcal{A}$. To do that, it will keep track of all the variables and sets appearing in the correctness game. In addition, the adversary $\mathcal{B}$ samples a random key K for the NSE scheme and constructs an empty list $\mathcal{N}_S$ that will represent a local copy of the same set appearing in the faithfulness game.

On $\mathcal{A}$'s $\textsc{Send}(A, M)$ oracle query, $\mathcal{B}$ queries its oracle F-$\textsc{Send}$ to receive the nonce N. Then it encrypts (N, A, M) under the key K to obtain C. It extends the list $\mathcal{T}$ with tuple (mn, A, M, C) and the list $\mathcal{N}_S$ with N, and increments mn. In the end, it returns C back to $\mathcal{A}$.

On $\mathcal{A}$'s $\textsc{Recv}(j)$ oracle query, $\mathcal{B}$ fetches corresponding (mn, A, M, C) from the list $\mathcal{T}$, sets the nonce N to be $\mathcal{N}_S[j]$, and M' to be M. It then queries F-$\textsc{Recv}(N)$ to obtain $(\mathrm{st}^{\mathrm{r}}_r, mn')$ back. The adversary $\mathcal{B}$ then returns (mn', M') back to $\mathcal{A}$. If the adversary $\mathcal{A}$ triggered the win flag in this $\textsc{Recv}(j)$ query it means that $mn' \neq j$ (since $M = M'$ by construction). By inspecting the code of F-$\textsc{Recv}(N)$ oracle, it can be seen that, in this case, a winning condition in the faithfulness game would be triggered as well. The faithfulness game calls the NonceSetPolicy algorithm and gets $\boldsymbol{W}$ back. If the queried nonce is not in $\boldsymbol{W}$, that constitutes a win since we know the nonce will be supported. Otherwise, queried nonce is in $\boldsymbol{W}$, and StUpdate returns mn'. This mn' is not equal to j in $\mathcal{B}$'s correctness game simulation, meaning that in the F-$\textsc{Recv}$ oracle, the condition $N \neq \mathcal{N}_S[mn']$ would be triggered. That conclusion follows from the fact that $N = \mathcal{N}_S[j]$ and $j \neq mn'$, and the property of nonce-set processing scheme algorithm NonceExtract always outputting unique nonces (thus nonces in $\mathcal{N}_S$ are all unique). Now, assume the adversary $\mathcal{A}$ did not trigger the win flag in this $\textsc{Recv}(j)$ query. Then $\mathcal{B}$ correctly simulates the response for $\mathcal{A}$, that is, returns the correct (mn', M). This follows from the fact that $N \in \boldsymbol{W}$, and with assumed NSE correctness, we know Recv algorithm would output (mn', M) for a ciphertext that was encrypted under the nonce $\mathcal{N}_S[j]$ and message M.

Reduction to the correctness of NSE*:* Under the assumption that NSP faithfully reproduces the support predicate supp, we know that if an adversary $\mathcal{A}$ wins in correctness game, it must be that for some query

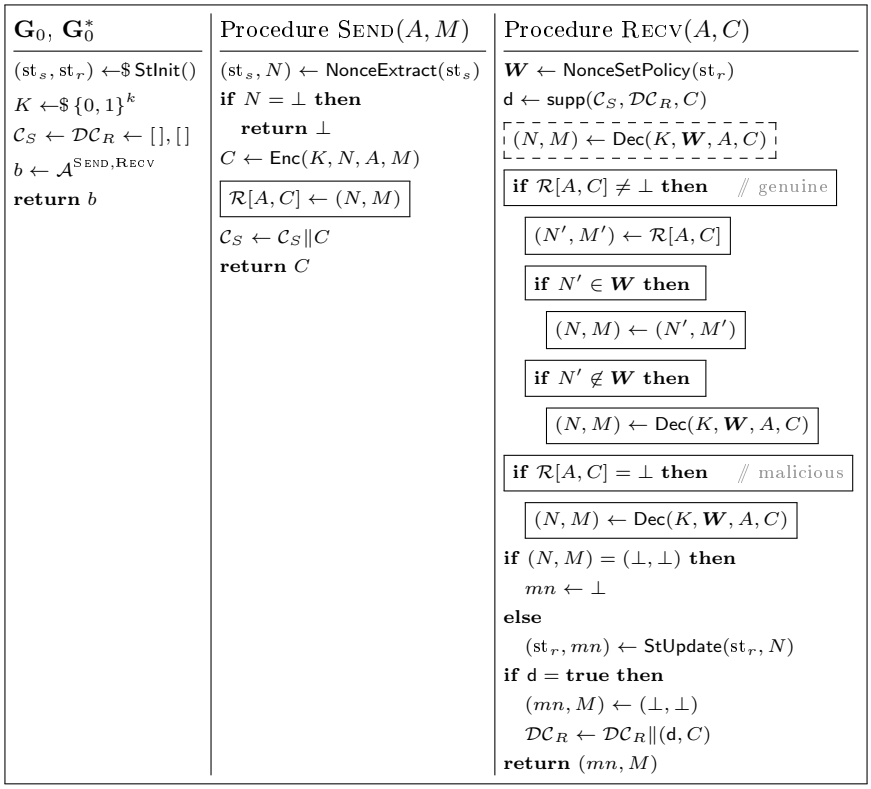

Fig. 30: The games $\mathbf{G}_0$ and $\mathbf{G}_0^*$ for the proof of Theorem 6. $\mathbf{G}_0$ does not contain the *solid-line* boxed code, and $\mathbf{G}_0^*$ does not contain the *dashed-line* boxed code.

$\textsc{Recv}(j)$, it happened that $M' \neq M$ (otherwise $mn' \neq mn$, and that reduces to breaking faithfulness, as we showed previously). The received message M' is part of the pair (N', M') that was output by the Dec algorithm on an input $(K, \boldsymbol{W}, A, C)$, where $\boldsymbol{W}$ was the output of $\mathsf{NonceSetPolicy}$ algorithm. Moreover, it holds that $N' = \mathcal{N}_S[j]$ and $N' \in \boldsymbol{W}$ (otherwise this would again reduce to breaking faithfulness). Since C was encrypted with the nonce $\mathcal{N}_S[j]$, we would have that $\mathtt{NSE}$ is not correct since it outputs a M' that was not encrypted initially. □

E.3 Proof of Theorem 6 (Nonce-set Channel Construction Security)

Proof. We prove the bound using the game hopping technique, and show the existence of a simulator S by constructing it through the course of the proof.

$\mathbf{G}_0$: This is the game $\mathsf{INT\text{-}SIM\text{-}CCA}$ with the bit b set to 0 (real world), and the code of channel construction algorithms (Send and Recv) embedded into it. Right away, we make a small change in $\textsc{Recv}$ oracle and move the support predicate calculation to the second line. The game is given in Fig. 30. We make changes to the game in order to obtain $\mathbf{G}_0^*$, given in same Fig.. We will use the table $\mathcal{R}$ to track which (N, A, M) triples were queried to the encryption algorithm and what the result was. We call the pairs (A, C) in $\mathcal{R}$ genuine. Otherwise, they are malicious. The games are identical. That is evident in $\textsc{Send}$ oracle. For $\textsc{Recv}$ oracle, suppose the queried pair (A, C) is genuine. If the $\mathsf{NonceSetPolicy}$ algorithm returns a nonce-set containing the nonce C was encrypted with, by the correctness of the $\mathtt{NSE}$ scheme, the decrypted message will be exactly M'. In this case, we see both games will continue the execution with the same (N', M'). If the nonce-set does not contain the nonce C was encrypted with, or if the (A, C) is a malicious pair, the result (N, M) will be a result of the nonce-set decryption algorithm in both $\mathbf{G}_0$ and $\mathbf{G}_0^*$.

$\mathbf{G}_1$: In the game $\mathbf{G}_1$, we replace the real nonce-set oracles with the ideal ones. We reduce $\mathcal{A}$'s distinguishing advantage to the nsAE security of $\mathtt{NSE}$. Let $\mathcal{B}$ be an adversary playing nsAE game of $\mathtt{NSE}$. The adversary $\mathcal{B}$ runs $\mathcal{A}$ and simulates the games $\mathbf{G}_0^*$ and $\mathbf{G}_1$ to it. The adversary $\mathcal{B}$ will locally generate sending sender and receiver states st_s and st_r, that it needs in order to simulate the games for $\mathcal{A}$. It

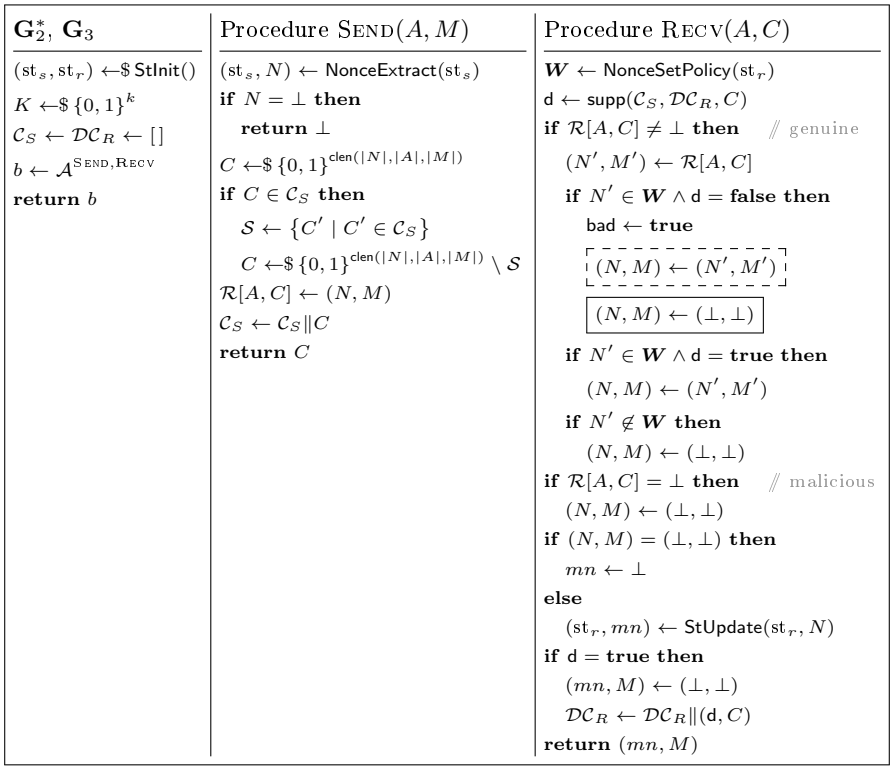

Fig. 31: The games $\mathbf{G}_2^*$ and $\mathbf{G}_3$ for the proof of Theorem 6. $\mathbf{G}_2^*$ does not contain the *solid-line* boxed code, and $\mathbf{G}_3$ does not contain the *dashed-line* boxed code.

will also track the sets $\mathcal{C}_S$ and $\mathcal{DC}_R$ to be able to calculate support predicate decisions itself. When an encryption or decryption oracle query in $\mathbf{G}_0^*$ or $\mathbf{G}_1$ is due, $\mathcal{B}$ calls its oracles. If $\mathcal{B}$ is playing the real game, it correctly simulates the game $\mathbf{G}_0^*$. Otherwise, it correctly simulates the game $\mathbf{G}_1$. Therefore, it holds that

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_0^*} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_1} \Rightarrow 1\right]\right| \leq \mathbf{Adv}^{\mathrm{nsae}}_{\mathtt{NSE}}(\mathcal{B}). \tag{19}$$

$\mathbf{G}_2$: In the game $\mathbf{G}_2$, we remove the possibility of collisions in ciphertexts (output by $\textsc{Send}$ oracle) happening. We arrange this in order to have unique ciphertexts in the set $\mathcal{C}_S$. That way, in the next game hop where we construct a reduction to the NSP faithfulness, we can have an equivalency between the support predicate in FAITHFUL game, that has nonces as unique identifiers, and support predicate in games $\mathbf{G}_2$ and $\mathbf{G}_3$, that has ciphertexts as unique identifiers.

Therefore, in the $\textsc{Send}$ oracle, we resample the ciphertext if it appeared already. The distinguishing bound between $\mathbf{G}_1$ and $\mathbf{G}_2$ then reduces to a birthday bound. It holds that

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_1} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_2} \Rightarrow 1\right]\right| \leq \frac{q_{\mathrm{s}}(q_{\mathrm{s}} - 1)}{2^{\ell+1}}, \tag{20}$$

where ℓ is the length of the shortest ciphertext C in the channel's ciphertext space.

$\mathbf{G}_3$: In order to arrive at the game $\mathbf{G}_3$, we first rewrite part of the code in $\textsc{Recv}$ oracle in $\mathbf{G}_2$ to get the equivalent game $\mathbf{G}_2^*$. We expand the check if $N' \in \boldsymbol{W}$ to two separate subcases, one where the support predicate decision was **false** and the other where the support predicate decision was **true**. The game $\mathbf{G}_2^*$, together with $\mathbf{G}_3$, is given in Fig. 31. The difference between $\mathbf{G}_2^*$ and $\mathbf{G}_3$ occurs when for a genuine ciphertext, $N' \in \boldsymbol{W}$ and $\mathsf{d} =$ **false**. In this case, we set N and M to $\perp$ in $\mathbf{G}_3$. We construct an adversary $\mathcal{D}$, playing the faithfulness game of NSP, that runs $\mathcal{A}$ and simulates the games $\mathbf{G}_2^*$ and $\mathbf{G}_3$ to it. $\mathcal{D}$ works as follows.

On $\mathcal{A}$'s $\textsc{Send}(A, M)$ query, it queries F-$\textsc{Send}$ oracle to receive the nonce N. Then it continues executing the rest of the code in $\textsc{Send}$ oracle and returns the resulting C to $\mathcal{A}$.

On $\mathcal{A}$'s $\textsc{Recv}(A, C)$ query, $\mathcal{D}$ executes the first two lines of code in the games to receive the nonce set $\boldsymbol{W}$ and support predicate decision d. Then it checks if an entry in the table $\mathcal{R}[A, C]$ exists. Here we differentiate two possible cases:

1. $\mathcal{R}[A,C] \neq \perp$. The adversary $\mathcal{D}$ reads (N', M') from the table $\mathcal{R}$. If N' is not in the set $\boldsymbol{W}$, $\mathcal{D}$ assigns $(\perp, \perp)$ to (N, M). Otherwise, the adversary $\mathcal{D}$ queries F-RECV(N') and receives a pair (st_r, mn) back. If $mn = \perp$, it assigns $(\perp, \perp)$ to (N, M), else, it assigns (N', M') to (N, M). Finally, the adversary $\mathcal{D}$ checks if the flag bad would be set[5]. If it would $\mathcal{D}$ aborts, otherwise it continues.

2. $\mathcal{R}[A,C] = \perp$. The adversary $\mathcal{D}$ assigns $(\perp, \perp)$ to the pair (N, M).

From there on, $\mathcal{D}$ finishes executing the RECV oracle code in the games as specified.

We claim the adversary $\mathcal{D}$ simulates the games for $\mathcal{A}$ correctly unless the event bad happens. By inspecting the code of RECV oracle, and the description of the adversary $\mathcal{D}$, it can be seen that the support predicates in FAITHFUL game and the games $\mathbf{G}_2^*$ and $\mathbf{G}_3$ will be in-sync during the whole simulation. In addition, the channel states will be in-sync as well. We see that the games $\mathbf{G}_2^*$ and $\mathbf{G}_3$ are identical-until-bad hence it holds that

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_2^*} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_3} \Rightarrow 1\right]\right| \leq \Pr[\mathsf{bad}].$$

If the event bad happens on $\mathcal{A}$'s receiving query to $\mathcal{D}$, it means $\mathcal{D}$ triggered the win condition in its own FAITHFUL game with the corresponding query to F-RECV. The flag bad means the nonce set contained the nonce used to encrypt the queried ciphertext, but the ciphertext was not supported. Furthermore, since the support predicates are in-sync, the condition

$$\mathsf{d} = \textbf{false} \ \wedge \ N' \in \boldsymbol{W}$$

would evaluate to **true**, which means the win flag in FAITHFUL would be set to **true**. Therefore, it holds that

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_2} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_3} \Rightarrow 1\right]\right| \leq \Pr[\mathsf{bad}] = \mathbf{Adv}_{\mathtt{NSP}}^{\mathrm{faithful}(\mathsf{supp})}(\mathcal{D}). \tag{21}$$

$\mathbf{G}_4$: This is the game INT-SIM-CCA with the bit set to 0 (ideal world). In comparison to the game $\mathbf{G}_3$, we make a change in SEND oracle by "switching back" to sampling random ciphertexts—the oracle does not resample C anymore if it already appeared in $\mathcal{C}_S$. It follows that

$$\left|\Pr\left[\mathcal{A}^{\mathbf{G}_3} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathbf{G}_4} \Rightarrow 1\right]\right| \leq \frac{q_{\mathrm{s}}(q_{\mathrm{s}} - 1)}{2^{\ell+1}}. \tag{22}$$

We can then replace the SEND oracle with a simple stateless simulator S defined as follows.

Simulator $\mathsf{S}(A, p)$
$s \leftarrow \mathsf{clen}(t, \lvert A\rvert, p)$
$C \leftarrow\!\!\$\ \{0,1\}^s$
return C

The clen is the ciphertext length function of NSE, t is the fixed length of nonces from the nonce space, and p is the message length. The RECV oracle is always returning $(\perp, \perp)$, same as the RECV oracle in the ideal world of INT-SIM-CCA. Therefore, the game $\mathbf{G}_4$ is equivalent to the ideal world of INT-SIM-CCA game.

Combining the inequalities (19), (20), (21) and (22) we achieve the bound in the theorem statement. □

E.4 Instantiating the Nonce-Set Processing Algorithms.

In Fig. 32 we present $\mathtt{NSP}_{\mathrm{I}}$, one possible concretization of nonce-set processing scheme. The nonce will consist of a message number masked with a nonce-randomizer IV, appended with two 0 bits (expressing the two reserved 0 bits appearing in the QUIC header) and a padding. The value t is the length of the nonces in the underlying NSE scheme. We show that $\mathtt{NSP}_{\mathrm{I}}$ is faithful with respect to the anti-replay and reordering support predicate $\mathsf{supp}_{\mathrm{rr}[\mathrm{w}_r, \mathrm{w}_f]}$ that was presented previously in Fig. 10.

Claim. The nonce-set processing scheme $\mathtt{NSP}_{\mathrm{I}}$ faithfully reproduces the support predicate $\mathsf{supp}_{\mathrm{rr}[\mathrm{w}_r, \mathrm{w}_f]}$.

[5] By checking if $N' \in \boldsymbol{W}$ and if $\mathsf{d} = \textbf{false}$.

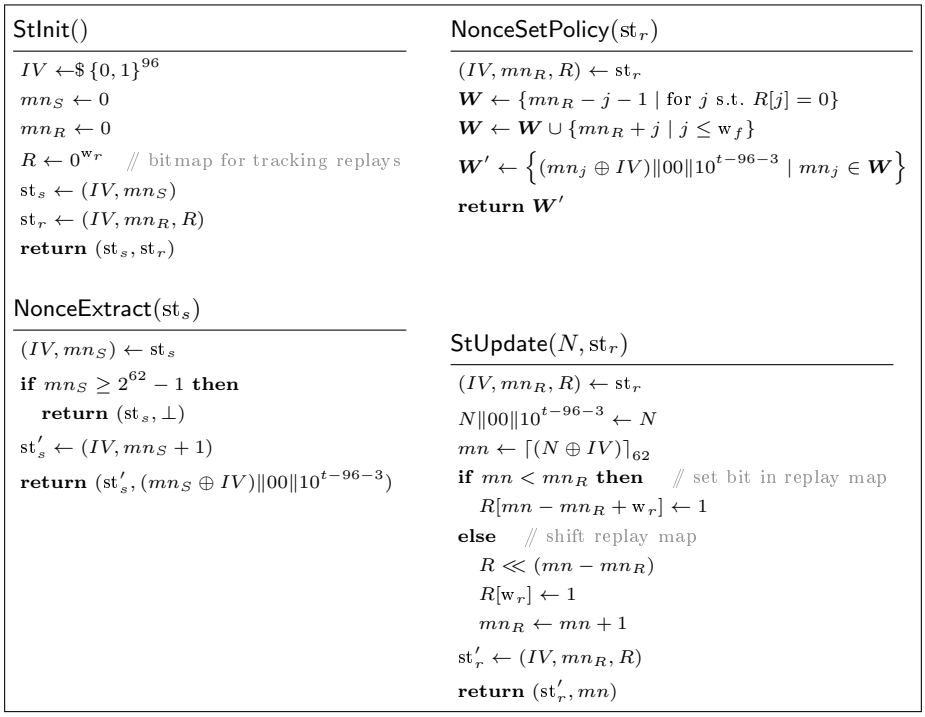

Fig. 32: The tuple $\mathtt{NSP}_\mathrm{I} = (\mathsf{StInit}, \mathsf{NonceExtract}, \mathsf{NonceSetPolicy}, \mathsf{StUpdate})$ for a concrete realization of an order-resilient secure channel. t is length of the nonce.

Proof. We prove the claim by contradiction. Suppose an adversary $\mathcal{A}$ triggers the win flag in FAITHFUL. Then for some $\mathcal{A}$'s query F-RECV(N) one of the following two holds.

1. $\mathsf{d} = \textbf{false} \wedge N \in \boldsymbol{W}$. By inspecting the code of the support predicate $\mathsf{supp}_{\mathrm{rr}[\mathrm{w}_r, \mathrm{w}_f]}$ given in Fig. 10, we differentiate three cases in which the predicate returns **false**. First one, when $N \notin \mathcal{N}_S$, cannot occur since faithfulness game requires that $N \in \mathcal{N}_S$. Second one, when the index of the nonce is less than $\mathsf{next} - \mathrm{w}_r$ or greater than $\mathsf{next} + \mathrm{w}_f$. In this case, we see by inspecting the NonceSetPolicy algorithm that such nonce would not be added to $\boldsymbol{W}$. Similar holds for the third case when the support predicate returns **false** because the nonce is replayed. The tracking of the replays in bitmap R would not allow the replayed nonce to be added in $\boldsymbol{W}$. Therefore, it cannot hold that $\mathsf{d} = \textbf{false}$ and $N \in \boldsymbol{W}$.
2. $\mathsf{d} = \textbf{true} \wedge (N \notin \boldsymbol{W} \vee N \neq \mathcal{N}_S[mn])$. The support predicate decision being **true** means that $N \in \mathcal{N}_S[mn]$, index of the nonce is inside the window $[\mathrm{w}_r, \mathrm{w}_f]$ around the next expected index, and the nonce has not been deemed supported before. By inspecting the NonceSetPolicy algorithm, one can see that in this case, the nonce N will be included in the nonce set $\boldsymbol{W}$. Moreover, from the StUpdate algorithm, it can be seen that the extraction of the nonce index (message number) is correct. That is, the nonce is correctly decoded into the respective message number. Thus, it holds that $N = \mathcal{N}_S[mn]$. The previous findings therefore contradict the starting assumption.

This shows that no adversary can win the faithfulness game against $\mathtt{NSP}_\mathrm{I}$ with respect to $\mathsf{supp}_{\mathrm{rr}[\mathrm{w}_r, \mathrm{w}_f]}$. □

The size of maximum set the algorithm NonceSetPolicy can output is $\mathrm{w}_r + \mathrm{w}_f$. Taking into account the values of reasonably-sized replay and reordering windows that we think could be employed in practice, the size of the nonce-set would not significantly influence the security of the secure channel.