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: Rugged Pseudorandom Permutations · UIV · Authenticate with Nonce · QUIC · DTLS · Tweakable Ciphers

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 [31], 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 [32]. 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 [26].

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

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

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 [34] 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 [32] 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.

1.1 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 [34] 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 [18] 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 [18] 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

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

such that for any $(K, T) \in \mathcal{K} \times \mathcal{T}$ the mapping $\widetilde{\mathsf{E}}\mathsf{E}(K, T, \cdot)$ 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 $\widetilde{\mathsf{E}}\mathsf{E}_K(T, \cdot)$ as shorthand for $\widetilde{\mathsf{E}}\mathsf{E}(K, T, \cdot)$ and $\widetilde{\mathsf{E}}\mathsf{E}_K^{-1}(T, \cdot)$ 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 $|\widetilde{\mathsf{E}}\mathsf{E}_K(T, X)| = |X|$. We also refer to $\widetilde{\mathsf{E}}\mathsf{E}$ and $\widetilde{\mathsf{E}}\mathsf{E}^{-1}$ 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 $\widetilde{\mathsf{E}}\mathsf{E}$ to denote a variable-input-length tweakable cipher, which may itself be constructed from an underlying tweakable blockcipher $\widetilde{\mathsf{E}}$.

Security. The typical security requirement for tweakable ciphers is the well-known (SPRP) notion. The formal definition can be found in the full version [16].

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

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

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

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

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

The special symbol $\perp$ 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* [26]. 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 $(K,N,H,C) \in \mathcal{K} \times \mathcal{N} \times \mathcal{H} \times \mathcal{C}$

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

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 $\perp(\cdot,\cdot,\cdot)$. 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*

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

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*

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

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*

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

Nonce-Hiding AEAD. In the full version of this paper [16] we cover the syntax of nonce-hiding AEAD and how the security definitions covered so far adapt 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, 34]. 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:

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

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. Furthemore, 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

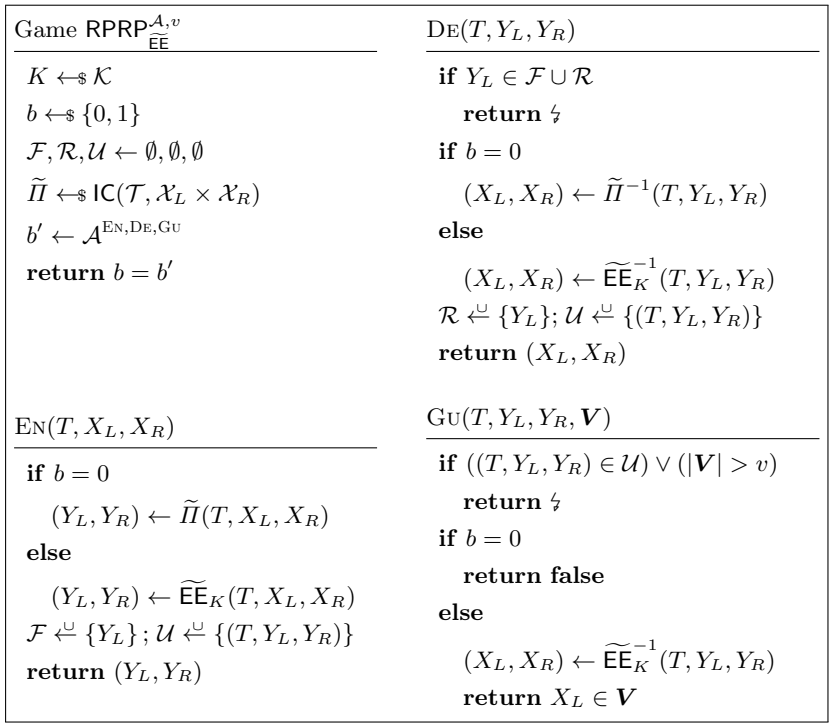

Fig. 2: The game used to define RPRP security for a tweakable cipher $\widetilde{\mathsf{EE}}$.

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.

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*

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

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 [34] 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 the full version of this paper [16].

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 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.*

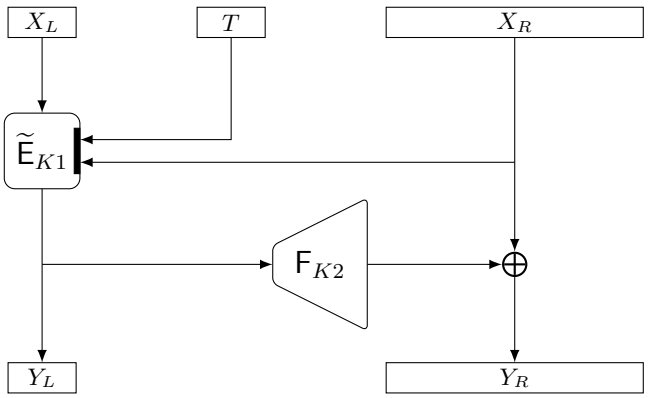

Fig. 4: Graphical representation of the `UIV` enciphering algorithm.

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 [24] using a blockcipher like AES and an Almost XOR-Universal (AXU) hash function like POLYVAL [20]. Alternatively one can use an off-the-shelf 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 [25].

As for the VOL-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. Alternatively, if one is using an off-the-shelf tweakable blockcipher, the VOL-PRF can be instantiated using the Counter-in-Tweak mode of operation [28], circumventing this issue entirely.

Notably, UIV can be fully instantiated from AES and POLYVAL, using LRW2 and counter mode, which benefit from the native instruction sets on many modern-day processors. The corresponding instantiation, GCM-UIV, shares many similarities with GCM-SIV [20] (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 [34]. 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. 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 and its proof is presented in the full version of this paper [16].

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*

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

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

$\mathsf{Enc}_K(N, H, M)$	$\mathsf{Dec}_K(N, H, C_1, C_2)$
$T \leftarrow \langle N, H \rangle$	$T \leftarrow \langle N, H \rangle$
$Z \leftarrow \mathsf{Func}_2(N, H)$	$Z \leftarrow \mathsf{Func}_2(N, H)$
$(C_1, C_2) \leftarrow \widetilde{\mathsf{EE}}_K(T, Z, M)$	$(Z', M') \leftarrow \widetilde{\mathsf{EE}}_K^{-1}(T, C_1, C_2)$
return (C_1, C_2)	**if** $Z' = Z$ **then**
	return M'
	else
	return $\perp$

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

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 and its proof can be found in the full version of this paper [16].

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*

$\mathsf{Enc}_K(N,H,M)$	$\mathsf{Dec}_K(N,H,C_1,C_2)$	$\Lambda_K(N,H,C_1,C_2)$
$T \leftarrow \langle N,H\rangle$	$T \leftarrow \langle N,H\rangle$	$T \leftarrow \langle N,H\rangle$
$Z \leftarrow \mathsf{Func}_3(N,H,M)$	$(Z',M') \leftarrow \widetilde{\mathsf{EE}}_K(T,C_1,C_2)$	$(Z',M') \leftarrow \widetilde{\mathsf{EE}}_K(T,C_1,C_2)$
$(C_1,C_2) \leftarrow \widetilde{\mathsf{EE}}_K^{-1}(T,Z,M)$	$Z \leftarrow \mathsf{Func}_3(N,H,M')$	$Z \leftarrow \mathsf{Func}_3(N,H,M')$
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.

(δ,t)-collision resistant deterministic function. Then there exists a leakage simulator S*, such that for any adversary* $\mathcal{A}$ *making* $q_e \leq 2^{n-1}$ *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*

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

The adversary $\mathcal{B}$ *makes* q_e *queries to* En *oracle,* q_d+q_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_d+q_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 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 provided in the full version of this paper [16]. 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 [8,9,21,34], 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 [34] 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

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

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

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

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*

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

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

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

$\mathsf{Enc}_K(N, H, M)$	$\mathsf{Dec}_K(\boldsymbol{W}, H, C_1, C_2)$
$l \leftarrow \lvert M \rvert$	$(X_L, X_R) \leftarrow \widetilde{\mathsf{EE}}_K^{-1}(H, C_1, C_2)$
$(C_1, C_2) \leftarrow \widetilde{\mathsf{EE}}_K(H, N \Vert \lfloor M \rfloor_{n-t}, \lceil M \rceil_{l-n+t})$	$N' \leftarrow \lfloor X_L \rfloor_t$
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. 7: 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 $t \leq n$.

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 described in the full version of this paper [16]. 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.

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. 7. 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 the full version of this paper [16].

Theorem 4. *Let* AwN *be the nonce-set AEAD scheme defined in Fig. 7, 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, 35] and DTLS [29,30], which operate over UDP, are two prime examples of order-resilient secure channels. Two recent works [17, 18] have analyzed the security of QUIC. Here we will follow in large part the formal security model of Fischlin, Günther, and Janson [18] which builds on and improves over prior works [6, 12, 33] 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 [33] 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 [18] to reflect the privacy requirement to conceal message numbers from eavesdroppers. In contrast, in [18] 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 [18] 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 [18] 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 [18] the prior models by [12,33] 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 [33], 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.

In conformance with [18], any ciphertext not in $\mathcal{C}_S$ must not be supported. However, whereas in [18] 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 [18]. 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 [18] 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, reflecting the required functionality of a typical real-world protocol, can be found in the full version of this paper [16]

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

Game $\mathsf{CORR}^{\mathcal{A}}_{\mathtt{Ch},\mathsf{supp}}$	Procedure $\textsc{Send}(A, M)$	Procedure $\textsc{Recv}(j)$
$(\mathrm{st}_s, \mathrm{st}_r) \leftarrow\!\!\$\ \mathtt{Ch.Init}()$	$(\mathrm{st}_s, C) \leftarrow\!\!\$\ \mathtt{Ch.Send}(\mathrm{st}_s, A, M)$	**if** $j > \lvert\mathcal{T}\rvert$ **then**
$\mathcal{DC}_R, \mathcal{C}_S, \mathcal{T} \leftarrow [\,], [\,], [\,]$	$\mathcal{C}_S \leftarrow \mathcal{C}_S \Vert C$	**return** $\lightning$
$mn \leftarrow 0$	$\mathcal{T} \leftarrow \mathcal{T} \Vert (mn, A, M, C)$	$(mn, A, M, C) \leftarrow \mathcal{T}[j]$
$\mathsf{win} \leftarrow$ **false**	$mn \leftarrow mn + 1$	$\mathsf{d} \leftarrow \mathsf{supp}(\mathcal{C}_S, \mathcal{DC}_R, C)$
$\mathcal{A}^{\textsc{Send},\textsc{Recv}}$	**return** C	**if** $\mathsf{d} =$ **false then**
return win		**return** $\lightning$
		$(\mathrm{st}_r, mn', M') \leftarrow \mathtt{Ch.Recv}(\mathrm{st}_r, A, C)$
		if $mn' \neq mn \vee M' \neq M$ **then**
		$\mathsf{win} \leftarrow$ **true**
		$\mathcal{DC}_R \leftarrow \mathcal{DC}_R \Vert (\mathsf{d}, C)$
		return (mn', M')

Fig. 8: The game CORR used to define channel correctness.

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. 8.

Definition 7 (Channel Correctness). *A channel* 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 does not capture such a scenario as it considers only honestly-generated ciphertexts. Accordingly, [18] introduced the notion of *robustness* to capture this stronger requirement.

Robustness is formally defined through the ROB game located in the full version of this paper [16]. 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 [18], without robustness, and adapted with some of the ideas from simulatable channels in [14]. Security is defined via the indistinguishability game $\mathsf{INT\text{-}SIM\text{-}CCA}$ shown in Fig. 9. 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,* $\mathtt{Ch}$ *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 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 [18]. 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

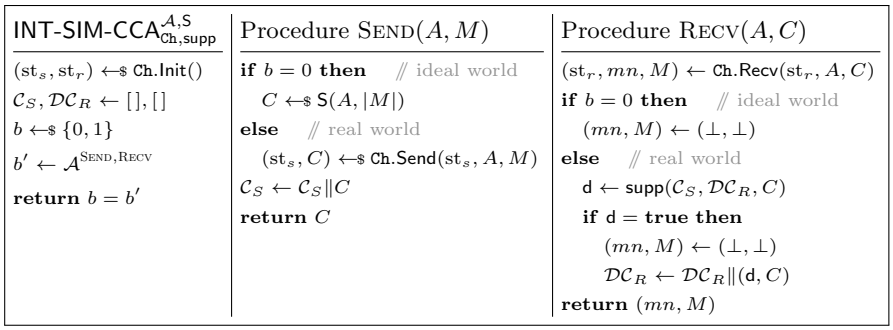

Fig. 9: The INT-SIM-CCA game used to define channel security.

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.

Game $\mathsf{FAITHFUL}^{\mathcal{A}}_{\mathtt{NSP},\mathsf{supp}}$	Procedure F-SEND()	Procedure F-RECV(N)

$(\mathrm{st}_s, \mathrm{st}_r) \xleftarrow{\$} \mathsf{StInit}()$
$\mathcal{N}_S, \mathcal{DC}_R \leftarrow [\,], [\,]$
$\mathsf{win} \leftarrow \mathbf{false}$
$\mathcal{A}^{(\mathrm{st}_s,\mathrm{st}_r),\text{F-SEND},\text{F-RECV}}$
return win
	$(\mathrm{st}_s, N) \leftarrow \mathsf{NonceExtract}(\mathrm{st}_s)$
	$\mathcal{N}_S \leftarrow \mathcal{N}_S \| N$
	return N
		if $N \notin \mathcal{N}_S$ **then**
		  return $\lightning$
		$\boldsymbol{W} \leftarrow \mathsf{NonceSetPolicy}(\mathrm{st}_r)$
		if $N \in \boldsymbol{W}$ **then**
		  $(\mathrm{st}_r, mn) \leftarrow \mathsf{StUpdate}(\mathrm{st}_r, N)$
		else
		  $mn \leftarrow \bot$
		$\mathsf{d} \leftarrow \mathsf{supp}(\mathcal{N}_S, \mathcal{DC}_R, N)$
		$\mathcal{DC}_R \leftarrow \mathcal{DC}_R \| (\mathsf{d}, N)$
		if $\mathsf{d} = \mathbf{true} \wedge$
		  $(N \notin \boldsymbol{W} \vee N \neq \mathcal{N}_S[mn])$ **then**
		  $\mathsf{win} \leftarrow \mathbf{true}$
		if $\mathsf{d} = \mathbf{false} \wedge N \in \boldsymbol{W}$ **then**
		  $\mathsf{win} \leftarrow \mathbf{true}$
		return (st_r, mn)

Fig. 10: The game FAITHFUL used to define faithfulness for a tuple of nonce set processing algorithms.

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. 10. 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.

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 NSE =

$\mathsf{Init}()$	$\mathsf{Send}(\mathrm{stk}_s, A, M)$	$\mathsf{Recv}(\mathrm{stk}_r, A, C)$
$(\mathrm{st}_s, \mathrm{st}_r) \leftarrow\!\$\ \mathsf{StInit}()$	$(\mathrm{st}_s, K) \leftarrow \mathrm{stk}_s$	$(\mathrm{st}_r, K) \leftarrow \mathrm{stk}_r$
$K \leftarrow\!\$\ \{0,1\}^k$	$(\mathrm{st}'_s, N) \leftarrow \mathsf{NonceExtract}(\mathrm{st}_s)$	$\boldsymbol{W} \leftarrow \mathsf{NonceSetPolicy}(\mathrm{st}_r)$
$\mathrm{stk}_s \leftarrow (\mathrm{st}_s, K)$	**if** $N = \bot$ **then**	$(N, M) \leftarrow \mathsf{Dec}(K, \boldsymbol{W}, A, C)$
$\mathrm{stk}_r \leftarrow (\mathrm{st}_r, K)$	**return** $(\mathrm{st}'_s, \bot)$	**if** $(N, M) = (\bot, \bot)$ **then**
return $(\mathrm{stk}_s, \mathrm{stk}_r)$	$C \leftarrow \mathsf{Enc}(K, N, A, M)$	$mn \leftarrow \bot$
	$\mathrm{stk}'_s \leftarrow (\mathrm{st}'_s, K)$	**else**
	return (stk'_s, C)	$(\mathrm{st}'_r, mn) \leftarrow \mathsf{StUpdate}(\mathrm{st}_r, N)$
		$\mathrm{stk}'_r \leftarrow (\mathrm{st}'_r, K)$
		return (stk'_r, mn, M)

Fig. 11: 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.

$(\mathsf{Enc}, \mathsf{Dec})$ and a nonce set processing scheme $\mathtt{NSP} = (\mathsf{StInit}, \mathsf{NonceExtract}, \mathsf{NonceSetPolicy}, \mathsf{StUpdate})$ is presented in Fig. 11.

Channel Correctness. The proof of correctness for this generic construction is provided in the full version of this paper [16].

Theorem 5. *If the nonce-set AEAD scheme* $\mathtt{NSE}$ *is correct and the nonce set processing scheme* $\mathtt{NSP}$ *faithfully reproduces the support predicate* supp*, then the channel construction* $\mathtt{Ch}_{\mathrm{NS}}$ *presented in Fig. 11 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 $\mathtt{NSP}$ guarantees that only the nonces used to generate supported ciphertexts will be included in the nonce set. Then, by the nsAE security of $\mathtt{NSE}$, 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 ($\mathsf{StUpdate}$ is not called) when decryption fails.

Channel Security. The security of $\mathtt{Ch}_{\mathrm{NS}}$ is formally stated in the following theorem, the proof of which is presented in the full version of this paper [16].

Theorem 6 (Security of $\mathtt{Ch}_{\mathrm{NS}}$). *Let* $\mathtt{Ch}_{\mathrm{NS}}$ *be the generic channel construction described in Fig. 11, composed from a nonce-set AEAD scheme* $\mathtt{NSE}$ *with associated ciphertext space* $\{0,1\}^{\geq \ell}$ *and a nonce set processing scheme* $\mathtt{NSP}$*. 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 $\mathcal{A}$.

Acknowledgments

We are grateful to Alessandro Melloni, Jean-Pierre Münch, and Martijn Stam for their collaboration in developing the RPRP security definition and other helpful discussions. We also thank the anonymous CRYPTO 2022 reviewers for their thorough reading and constructive comments. This research was supported by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

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 [19], 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 [27], 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 [27], 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. J. P. Degabriele and V. Karadžić. Overloading the nonce: Rugged PRPs, nonce-set AEAD, and order-resilient channels. Cryptology ePrint Archive, Paper 2022/817, 2022. https://eprint.iacr.org/2022/817.
17. 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.
18. 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.
19. J. Groth, editor. *15th IMA International Conference on Cryptography and Coding*, volume 9496 of *LNCS*. Springer, Heidelberg, Dec. 2015.
20. 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.
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. M. Liskov, R. L. Rivest, and D. Wagner. Tweakable block ciphers. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 31–46. Springer, Heidelberg, Aug. 2002.
25. K. Minematsu and T. Iwata. Tweak-length extension for tweakable blockciphers. In Groth [19], pages 77–93.
26. 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.
27. T. Okamoto, editor. *ASIACRYPT 2000*, volume 1976 of *LNCS*. Springer, Heidelberg, Dec. 2000.

28. T. Peyrin and Y. Seurin. Counter-in-tweak: Authenticated encryption modes for tweakable block ciphers. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 33–63. Springer, Heidelberg, Aug. 2016.
29. E. Rescorla and N. Modadugu. *The Datagram Transport Layer Security Version 1.2*, 2012. https://datatracker.ietf.org/doc/html/rfc6347.
30. 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/.
31. 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.
32. 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.
33. 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.
34. 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.
35. M. Thomson and S. Turner. *RFC9001: Using TLS to Secure QUIC*, 2021. https://datatracker.ietf.org/doc/html/rfc9001.