Jammin' on the deck

Norica Bcuiei[1], Joan Daemen[1], Seth Hoffert, Gilles Van Assche[2], and Ronny Van Keer[2]

[1] Radboud University, Nijmegen, The Netherlands
[2] STMicroelectronics, Diegem, Belgium

Abstract. Currently, a vast majority of symmetric-key cryptographic schemes are built as block cipher modes. The block cipher is designed to be hard to distinguish from a random permutation and this is supported by cryptanalysis, while (good) modes can be proven secure if a random permutation takes the place of the block cipher. As such, block ciphers form an abstraction level that marks the border between cryptanalysis and security proofs. In this paper, we investigate a re-factored version of symmetric-key cryptography built not around the block ciphers but rather the deck function: a keyed function with arbitrary input and output length and incrementality properties. This allows for modes of use that are simpler to analyze and still very efficient thanks to the excellent performance of currently proposed deck functions. We focus on authenticated encryption (AE) modes with varying levels of robustness. Our modes have built-in support for sessions, but are also efficient without them. As a by-product, we define a new ideal model for AE dubbed the *jammin cipher*. Unlike the OAE2 security models, the jammin cipher is both a operational ideal scheme and a security reference, and addresses real-world use cases such as bi-directional communication and multi-key security.

Keywords: deck functions, authenticated encryption, wide block cipher, modes of use, ideal model

1 Introduction

Currently, a vast majority of symmetric-key cryptographic schemes are built as a mode of use of a block cipher. A block cipher is governed by a secret key and transforms an input block of fixed length into an output block of the same length, and as such its functionality is rather limited. However, the existence of powerful modes of use really unleashes the power of block ciphers: Combining them allows building cryptographic schemes for encryption, authentication and authenticated encryption of messages consisting of arbitrary-length plaintext and associated data. Block ciphers have even been used to build hash functions.

1.1 Moving to deck functions

Need for PRF security Modes of use usually come with a security guarantee: Assuming the underlying block cipher satisfies some security criterion, the cryptographic scheme can be proven secure. Often, this criterion is that the block

cipher, when keyed with a uniformly chosen key unknown to the adversary, is hard to distinguish from a random permutation; this is known as the pseudorandom permutation (PRP) security of a block cipher, in the case that an adversary is only allowed to query the block cipher in the forward direction, otherwise it is called strong PRP (SPRP) security. The PRP and SPRP security notions have become so accepted that they are referred to as the *standard model*. Thanks to this split in block ciphers and modes, the assurance of such cryptographic schemes relies on public scrutiny of the block cipher with respect to its (S)PRP security.

The security guarantee of many modes hit the so-called birthday bound and that causes the security of block-cipher based modes to break down as soon as the data complexity reaches $2^{n/2}$, with n the block size. This accounts for the presence, or absence, of collisions in block cipher outputs, depending on the mode.

Hitting this birthday bound is due to the invertibility of the block cipher while most modern block cipher modes do not even use the inverse block cipher.[3] Such modes often rely on the keyed block cipher to behave like a random function rather than a permutation, e.g., see [33], and this is called *pseudorandom function* (PRF) security.

Variable-input and output lengths Block cipher modes parse variable-length inputs as fixed-length blocks. This often comes with considerable complexity, such as dealing with complete last blocks, that propagates to the security proofs. Modes would be simpler if the underlying primitive would *natively* support variable input and output lengths. Moreover, (S)PRP security makes little sense for a primitive with variable input and output lengths, and striving for good PRF security makes more sense.

Such primitives would be a good replacement of block ciphers as a focus point in symmetric key cryptography and they have actually been proposed by Daemen et al. under the name of *deck function* [15]. A construction for building deck functions is called *farfalle*, and the authors showcased an instance of farfalle based on Keccak-f called Kravatte with excellent performance, and later a second one called Xoofff improving on all aspects over Kravatte [8,15]. But *deck function* just specifies an interface and farfalle is not the only way to build a deck function, in the same way that there are multiple ways to build a block cipher: a wide design space is waiting to be explored!

Performance We focus on authenticated encryption (AE), i.e., schemes that simultaneously achieve confidentiality and authenticity [7,36]. Next to the simplicity of modes, performance is a clear and natural motivation for exploring AE using deck functions. For instance, Kravatte and Xoofff have excellent

[3] The input and output of a block cipher are often called plaintext and ciphertext, respectively. This may be correct for the ECB mode, but for the majority of today's modes, the input is not the plaintext and/or the output is not the ciphertext.

reported performance figures and outperform modes using the AES block cipher, sometimes even when the platform has hardware AES support [13]. Even if faster block ciphers can be built, security proofs of their modes rely on their (S)PRP security, and achieving a solid level of (S)PRP security comes at the price of a relatively large number of rounds. Building a variable-input-length function that targets PRF security using the same building blocks can be done more efficiently when the reductionist security argument is dropped. We illustrate this with two MAC functions: CMAC [34] with underlying block cipher AES-128 [18] and Pelican-MAC [19]: for long messages the former costs 10 AES rounds while the latter only 4 (unkeyed) AES rounds per 128-bit block of input. Despite the absence of a reductionist security proof, Pelican-MAC has maintained its security claim, very close to that of CMAC with AES-128, up to this day. A similar argument can be made for functions with variable-length output. Efficient deck functions support both a variable-length input and output and trade reductionist (S)PRP-based security proofs in for security based on cryptanalysis. Clearly, deck function-based cryptography seems like an alternative to block-ciphers that is worth exploring.

1.2 Processing sessions

Sequences of messages Today's applications for cryptography go beyond the encryption or authentication of individual messages. The processing of streams of data, with intermediate tags, and bi-directional communication are common use cases. To this end, we cover as well the traditional authenticated encryption of a single message, i.e., a plaintext-associated data pair, as the authenticated encryption of a sequence of messages. More specifically, we define a *session* as the process of authenticating a message in the context of previously sent ones within the sequence. By extension, we also speak of a session for a sequence of messages that are processed in this way. We envision session-supporting AE as a scheme holding a rolling state that accumulates the messages as they are processed.

There exist other techniques for dealing with a sequence of messages. First, *stateful* authenticated encryption (sAE) refers to a scheme that deals with reordering, replay and omission of messages [6,41]. In its most generic definition, it is parameterized with is a set of admissible message numbers. E.g., if the sender emits messages $(1, 2, 3, 4)$, would the receiver accept ciphertext messages arriving the order $(2, 1, 2, 4)$? In a sense, our concept of session is a special case of sAE where the only admissible sequence of ciphertexts is the original sequence.

Then, *online* authenticated encryption refers to the ability to decrypt on the fly with a bounded memory size, see Hoang et al. [25] (HRRV). A typical example would be the encryption of a large message (e.g., a movie) that is cut it into segments (e.g., chunks of a few seconds), each of these being authenticated. Our concept of session can implement such a use case, although with a diverging definition of a message: One *segment* in HRRV is treated as one *message* in our session. Yet, it is not in HRRV's philosophy to support bi-directional communications as it focuses on sending just one message, even if cut into segments.

If a unique identifier N of the message sequence is available, authenticated encrypting a sequence of messages can be approached in a rather simple way. We can simply encrypt messages under the diversifier (N, n), with n the message number within the sequence, since (N, n) is a nonce. As a rule, the receiver only decrypts and checks message (N, n) if all the previous messages $(N, n' < n)$ were correct.

Adding a rolling state that accumulates the messages as they are processed gives additional benefits. First, it provides further robustness. If a message is tampered with at any point in the session, the rest of the session becomes completely corruptedso if the attacker somehow tricks the implementation into continuing after a bad tag, everything will look like random noise. It forces the implementation to deal with a bad message and it becomes impossible to ignore it. With the previous approach, it was still possible to decrypt message $n + 1$ successfully even if message n was corrupted.

Second, the management of unique diversifiers becomes simpler and more natural. Uniqueness of the diversifier must be ensured at the level of sessions, as individual messages within the session are diversified at least by the number of messages received so far. If a key is bound to only one session, the need of session-level unique diversifiers even vanishes; this can happen, e.g., if the key is the result of a ephemeral Diffie-Hellman key exchange aimed at securing one particular session. Also, with diversifier-reuse-resistant modes, the rest of the session becomes perfectly diversified if any message in the session contains a diversifier.

Sessions and incrementality There is another reason for looking at sessions with a rolling state. Several symmetric cryptographic primitives and modes support *incrementality* properties, i.e., by keeping state, appending additional input or requesting further output does not require to re-process everything from the beginning. Note that this is also an explicitly required property of deck functions.

Incrementality comes in handy when defining session-supporting AE schemes: A formal definition specifies that any output of the scheme (keystream or authentication tag) depends on the entire sequence of messages received so far, while the implementation relies on the incrementality and a rolling state to process each message only once. A striking example is the duplex construction that provides an incremental interface to the sponge construction, on top of which it is fairly easy to build session-supporting AE [9].

We could say that the definition of sessions was influenced by the existence of incrementality properties, but in the end sessions and incrementality combine in an elegant and useful way.

1.3 Looking for an ideal model

Having set out our goal as to build AE modes on top of a deck function, we need to opt for an ideal model. An obvious choice would be the $(\$, \perp)$ model: The ciphertext looks random and any decryption attempt returns an error. However, this model is referential but not operational:

– *operational*: serves as an ideal scheme for use in higher-level protocols;
– *referential*: serves as an ideal-world model in distinguishability settings for modes or schemes, to prove, or claim, a distinguishing bound.

We aim for an ideal scheme that is both operational and referential. This allows using it in a higher-level protocol, prove that secure, and subsequently instantiate it with a concrete AE scheme; the security of the resulting protocol can then be quantified using the triangle inequality. Some definitions come as a pair of an operational and a referential model, but often with a security gap between the two. For instance, the referential deterministic AE (DAE) is paired with the operational pseudo-random injection (PRI) [40]. In the former encryption gives uniformly distributed ciphertext without replacement and is hence non-deterministic and decryption always returns an error.

The online authenticated encryption 2 (OAE2) security definitions from HRRV are the closest to what we try to achieve [25]. Specifically, OAE2 supports something very close to our sessions and covers streaming applications, where plaintexts and ciphertexts can be processed on the fly. However, they define not a single ideal-world scheme, but a set of three schemes, Ideal2A, Ideal2B and Rand2C. The former two are operational and define the same security concept, but have different interfaces, whereas the third, Rand2C, is referential-only and defines a different security concept. In particular, in Ideal2A and Ideal2B forgery is possible and in Rand2C forgery is impossible by construction. Interestingly, the security gap between Ideal2 and Rand2C is *larger than the one between the modes we define in this paper and our ideal scheme*, see Section 2.4.

1.4 Our contributions

Our two main contributions are an ideal model and a number of deck function modes, both for session-supporting and non-session-supporting authenticated encryption.

The ultimate ideal-world authenticated encryption scheme We define the *jammin cipher* that is at the same time deterministic, session-supporting, operational and referential. The designation ultimate means it achieves the highest security thinkable while behaving deterministically: Forgery is impossible, the cryptograms are as random as injectivity allows and equal inputs give equal outputs in the same context. Our ideal-world scheme supports sessions, although it naturally also covers non-session AE.

Besides combining operational and referential roles in a single scheme, the jammin cipher has several interesting features and compares favorably to OAE2:

– It can serve as a security reference for *both nonce-enforcing and nonce-misuse-resistant schemes*. For OAE2, variants like nOAE or dOAE must be used instead [25].
– It produces cryptograms whose *distribution is intuitive and is as random as allowed while leaving the possibility for decryption*. In contrast, the definition

of Ideal2A/B make use of a rather complex building block IdealOAE(τ), called uniformly sampled τ-expanding injective functions.
– It has *ciphertext expansion* as a parameter, required when dealing with schemes that have variable ciphertext expansion due to the use of block encryption.
– It addresses *multi-key security* by supporting multiple instances. While OAE2 focuses on single-key security, Hoang and Shen propose a generalization of nOAE called nOAE2 in the multi-key setting [26].
– It supports unwrap and wrap calls in any order, including *bi-directional communication*. Instead, an instance of Ideal2B can only encipher messages or decipher cryptograms but not both.

Deck function-based session-supporting authenticated encryption We introduce a number of modes based on deck functions, with different combinations of features, as summarized in Table 1. For the last four modes, we propose a unified approach of achieving several security goals via a Feistel network.

There exist generic modes for building session-supporting AE, like CHAIN and STREAM [25]. However, these build a secure session AE scheme using a secure conventional AE scheme whereas we build both using a deck function. Some other constructions are block-oriented, which is what we try to improve using deck functions [1, 5, 10, 21]. The authenticated streamwise on-line encryption (ASOE) construction explicitly avoids blocks, but it aims for a weaker security notion [42]. Note that Barbosa and Farshim point out that an indifferentiable AE can be realized via a 3-round Feistel network [4].

Mode	Section	Tolerates nonce misuse	Tolerates release of unverified plaintext	Minimal ciphertext expansion
Deck-PLAIN	4			✓
Deck-BO	5.1	✓		
Deck-BOREE	5.2	✓	✓	
Deck-JAMBO	5.3	✓		✓
Deck-JAMBOREE	5.4	✓	✓	✓

Table 1: Overview of our AE modes.

Note that the jammin cipher and the deck function-based modes are efficient for single message AE and can be used in sAE modes.

1.5 Conventions

The set of all bit strings is denoted $\mathbb{Z}_2^*$ and ϵ is the empty string. The length in bits of the string X is denoted $|X|$. The concatenation of two strings X, Y is denoted as $X||Y$ and their bitwise addition as $X + Y$, with the resulting string

having length $\min(|X|, |Y|)$. Bit string values are noted with a typewriter font, such as $\mathtt{01101}$. The repetition of a bit is noted in exponent, e.g., $\mathtt{0}^3 = \mathtt{000}$. In a sequence of m strings, we separate the individual strings with a semicolumn, i.e., $X^{(0)}; X^{(1)}; \ldots; X^{(m-1)}$. The set of all sequences of strings is denoted $(\mathbb{Z}_2^*)^*$ and $\varnothing$ is the sequence containing no strings at all. Similarly, the set of all sequences containing at least one string is denoted $(\mathbb{Z}_2^*)^+$. Finally, $\varnothing$ is the empty set and $\perp$ denotes an error code.

In this paper we perform security analysis in the *distinguishability framework* where one bounds the advantage of an adversary $\mathcal{A}$ in distinguishing a real-world system from an ideal-world system.

Definition 1. *Let $\mathcal{O}, \mathcal{P}$ be two collections of oracles with the same interface. The advantage of an adversary $\mathcal{A}$ in distinguishing $\mathcal{O}$ from $\mathcal{P}$ is defined as*

$$\Delta_{\mathcal{A}}(\mathcal{O}\ ;\ \mathcal{P}) = \left|\Pr\left(\mathcal{A}^{\mathcal{O}} \to 1\right) - \Pr\left(\mathcal{A}^{\mathcal{P}} \to 1\right)\right|.$$

Here $\mathcal{A}$ is an algorithm that returns 0 or 1.

If we can build a real-world system $\mathcal{P}$ that is hard to distinguish from the ideal-world system $\mathcal{O}$, then we can replace $\mathcal{O}$ by $\mathcal{P}$ in the protocol without sacrificing much security. Concretely, if we can prove an upper bound on the distinguishing advantage $\Delta_{\mathcal{A}}(\mathcal{O}\ ;\ \mathcal{P})$ for any adversary $\mathcal{A}$, the attack success probability increases by at most that bound.

1.6 Outline

In Section 2, we define the jammin cipher. In Section 3, we discuss deck functions and some of their basic applications. In Section 4 we define Deck-PLAIN, the simplest of our five AE modes. If using a strong deck function and on the condition that the encryption context is a nonce, Deck-PLAIN can be distinguished from the jammin cipher only through tag guessing. In Section 5, we introduce four modes that do not require the encryption context to be a nonce, with different properties.

2 The jammin cipher, an ideal-world AE scheme

We define the *jammin cipher* in Algorithm 1. We describe it in an object-oriented way, with *object instances* (or *instances* for short) held by the communicating parties. An instance belongs to a given party who initializes it with an object identifier ID. Such an identifier is the counterpart of a secret key in the real world: Encryption and decryption will work consistently only between instances initialized with the same identifier. This setup models independent pairs (or groups) that make use of the AE scheme simultaneously. For example, Alice and Bob may secure their communication each using instances that share the same identifier $\mathrm{ID}_{\text{Alice and Bob}}$, while Edward and Emma use instances initialized with $\mathrm{ID}_{\text{Edward and Emma}}$. We will informally call an *object* the set of instances

Algorithm 1 The jammin cipher $\mathcal{J}^{\mathrm{WrapExpand}(p)}$

Parameter: WrapExpand, a t-expanding function
Global variables: codebook initially set to $\perp$ for all, taboo initially set to *empty*

Instance constructor: init(ID)
return new instance inst with attribute inst.history = ID

Instance cloner: inst.clone()
return new instance inst′ with the history attribute copied from inst

Interface: inst.wrap(A, P) returns C
context $\leftarrow$ inst.history; A
if codebook(context; P) $= \perp$ **then**
  $\mathcal{C} = \mathbb{Z}_2^{\mathrm{WrapExpand}(|P|)} \setminus (\mathsf{codebook}(\mathsf{context}; *) \cup \mathsf{taboo}(\mathsf{context}))$
  if $\mathcal{C} = \varnothing$ **then return** $\perp$
  codebook(context; P) $\xleftarrow{\$}$ $\mathcal{C}$
inst.history $\leftarrow$ inst.history; A; P
return codebook(context; P)

Interface: inst.unwrap(A, C) returns P or $\perp$
context $\leftarrow$ inst.history; A
if $\exists! P$: codebook(context; P) $= C$ **then**
  inst.history $\leftarrow$ inst.history; A; P
  return P
else
  taboo(context) $\leftarrow C$
  return $\perp$

sharing the same object identifier. This way, all the instances of the same object have indistinguishable behavior, and this justifies that we collectively call them an object, whereas instances of different objects are completely independent.

Our scheme supports two functions: wrap and unwrap. With the wrap function the object computes a cryptogram C from a message that has a plaintext P and associated data A, both arbitrary bit strings. With the unwrap function the object computes the plaintext P from the cryptogram C and A again. The cryptogram C is the encryption of P for a given A.

The jammin cipher is parameterized with a function WrapExpand(p) that specifies the length of the cryptogram given the length p of the plaintext. Typical examples observed in AE schemes in the literature are WrapExpand$(p) = p + t$ with t some fixed length, e.g., 128 for stream encryption followed by a 128-bit tag. For OCB [39], we have WrapExpand$(p) = t \left\lceil \frac{p}{t} + 1 \right\rceil$ with t the block length of the cipher. Both are examples of *t-expanding* functions. For use with the jammin cipher, we require WrapExpand to satisfy this property, defined below.

Definition 2. *A function* $f\colon \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ *is* t-expanding *iff (i)* $\forall \ell > 0\colon f(\ell) > f(0)$ *and (ii)* $\forall \ell\colon f(\ell) \geq \ell + t$.

Property (i) is needed in some of the modes to distinguish authentication-only messages from others. Property (ii) allows us to use t as a security parameter: the advantage of distinguishing a real-world scheme from an ideal scheme will be lower bound by an expression in the number of queries multiplied by 2^{-t}.

When two parties communicate, they usually have more than one message to send to each other. And a message is often a response to a previous request, or in general its meaning is to be understood in the context of the previous messages. The jammin cipher is *stateful*, where the sequence of messages exchanged so far is tracked in the attribute history. Initialization sets this attribute to the object identifier and each wrap and (successful) unwrap appends a message (A, P). So history is a sequence with ID followed by zero, one or more messages (A, P).

A *session* is the process in which the history grows with the messages exchanged so far. The wrap and unwrap functions make the history act as associated data, so that a cryptogram authenticates not only the message (A, P) but also the sequence of messages exchanged so far. An important application of this are intermediate tags, which authenticate a long message in an incremental way.

Finally, a jammin cipher object can be cloned. This is the ideal world's equivalent of making a copy of the state of the cipher. This means the user can save the history and restart from it ad libitum.

2.1 Inner workings

The jammin cipher keeps track of all wrap queries in a global archive called codebook. This is a mapping from tuples $(\mathsf{history}; A; P)$ to a cryptogram or an error code. The data elements history and A together form the *context* for the encryption of P: In different contexts, the jammin cipher encrypts plaintexts independently. We write $\mathsf{context} \leftarrow \mathsf{history}; A$ as the context for encryption in a wrap call, or decryption in an unwrap call, is the history with A appended.

Initially, all the entries of codebook return an error. In the algorithm, the expression $\mathsf{codebook}(\mathsf{context}; P) \xleftarrow{\$} \mathcal{S}$ denotes the assignment of a random element chosen uniformly from $\mathcal{S}$ to the entry $\mathsf{codebook}(\mathsf{context}; P)$, and $\mathsf{codebook}(\mathsf{context}; *)$ denotes the set of the values of $\mathsf{codebook}(\mathsf{context}; P)$ over all P.

Similarly, the jammin cipher keeps track of invalid cryptograms in a global archive called taboo. This is a mapping from (decryption) contexts to a set of cryptograms. Initially taboo is empty and with each attempt at decryption of an invalid cryptogram, it adds the cryptogram to the set of the corresponding context $\mathsf{context} = \mathsf{history}; A$. The expression $\mathsf{taboo}(\mathsf{context}) \leftarrow C$ denotes the addition of C to $\mathsf{taboo}(\mathsf{context})$.

Cryptograms in codebook are never overwritten, as the only place where a cryptogram value is assigned to codebook is on line 12, under the condition that codebook previously contains $\perp$. This makes wrapping deterministic. Similarly, the jammin cipher will unwrap any ciphertext C to the same plaintext value in any given context, i.e., unwrapping is deterministic. This is formalized in the following property.

Proposition 1. *From* codebook *one always recovers at most one plaintext value:*

$$\forall(\mathsf{context}, C), |\{P : \mathsf{codebook}(\mathsf{context}; P) = C\}| \leq 1.$$

Proof. Let $C \in \mathcal{C}$ be the value that is added to $\mathsf{codebook}(\mathsf{context}; P)$ in line 12. If $P' \neq P$ was another plaintext value such that $\mathsf{codebook}(\mathsf{context}; P') = C$, then we would get a contradiction as $C \in \mathsf{codebook}(\mathsf{context}; *)$ and thus $C \notin \mathcal{C}$, proving the proposition. □

We see that in line 11, wrap may return an error and therefore exhibit non-ideal behavior. We will now prove that for reasonable ciphertext expansion this requires an excessive number of specific unsuccessful unwrap queries.

Proposition 2. *If* WrapExpand *is t-expanding with $t \geq 2$,* wrap *is successful unless there were at least 2^t unsuccessful unwrap queries with the same context.*

Proof. A necessary condition for an error to be returned is the following. There exists a context and a cryptogram length n such that the sum of the following two items is at least 2^n:

- the number of calls to $\mathsf{wrap}(A, P)$ with $\text{WrapExpand}(|P|) = n$,
- the number of unsuccessful calls to $\mathsf{unwrap}(A, C)$ with $|C| = n$.

This is because the cardinality of $\mathcal{C}$ in line 10 is at least 2^n minus the number of n-bit strings in $\mathsf{codebook}(\mathsf{context}; *)$ or in $\mathsf{taboo}(\mathsf{context})$.

First, let us consider the case where $n = \text{WrapExpand}(0) \geq t$ with $P = \epsilon$. Given that WrapExpand is t-expanding, only $\mathsf{taboo}(\mathsf{context})$ can exclude possible cryptograms from $\mathcal{C}$ on line 10. It is therefore necessary to have at least $2^n \geq 2^t$ unsuccessful calls to unwrap.

Then, say $n > \text{WrapExpand}(0)$. The number of plaintext values that wrap to ciphertexts of size n is limited to 2^{n-t+1}. The possible plaintext lengths p are such that $\text{WrapExpand}(p) = n$ but they must satisfy $p \leq n - t$. Summing over all such possible lengths, the number of distinct plaintext values is upper bounded by 2^{n-t+1}. For line 11 to return an error, it is therefore necessary to have at least $2^n - 2^{n-t+1}$ unsuccessful calls to unwrap. Since $n > t \geq 2$ this is lower bounded by 2^t. □

2.2 Properties

The jammin cipher enjoys the following properties:

Deterministic wrapping: In a given context, an object wraps equal messages (A, P) to equal cryptograms C. It achieves this by tracking the cryptograms in the codebook archive.

Injective wrapping: An object wraps messages with equal context and A and different P to different cryptograms. It achieves this by excluding cryptogram values that it returned in earlier wrap calls for the same context and A.

Random cryptograms: Except for determinism and injectivity, all cryptograms C are fully random.

Deterministic unwrapping: In a given context, an object unwraps equal cryptograms to equal responses. It achieves this by tracking in taboo cryptogram values that it returns an error to.

Correctness: Thanks to deterministic (un)wrapping and injective wrapping, one jammin cipher object correctly unwraps what another wrapped, whenever their contexts are equal.

Forgery-freeness: In a given context, an object will only unwrap successfully cryptograms C resulting from prior wrap calls in the same context.

2.3 Discussion

Deterministic AE leaks in the sense identical plaintexts map to identical cryptograms. In particular, if the possible plaintexts form a small set, an adversary can recover it from the cryptogram by wrapping all of them. This opens to a family of attacks such as the chosen-prefix secret-suffix (CPSS) attack [20, 25].

The countermeasure against these attacks is to make encryption *context-dependent*. If the user can ensure that the encryption context is unique per plaintext, equal plaintexts will give different cryptograms. Usually, this context is a message counter (e.g., in counter mode) or a (random) initial value (e.g., in CBC) and called a *nonce*. This naming is confusing when discussing use cases where the uniqueness of the data element cannot be guaranteed.

The jammin cipher does not enforce the encryption context to be a nonce, this is left up to the higher level protocol or use case.

The jammin cipher takes as encryption context the sequence of messages exchanged so far, including the associated data in the message containing the plaintext to be encrypted (in a message without plaintext, there is no encryption and hence no encryption context). The advantage of doing authenticated encryption in sessions is immediate as this reduces the requirement for global diversifiers of one per session rather than one per message. Session-level diversifiers may even be omitted unless communicating parties wish to start parallel threads or start afresh from the same shared key.

Definition 3. *We say that the* encryption context is a nonce *iff all wrap queries with non-empty plaintext have a different context* context.

In case of re-use of encryption context, the jammin cipher will leak equality of plaintexts given equal cryptograms obtained with equal encryption contexts, but nothing more. In some use cases this may be acceptable. For such use cases, the jammin cipher can serve as a security reference for modes or schemes. A proven upper bound on the distinguishing advantage between such a mode and the jammin cipher, proves that leakage is limited to equal plaintexts and encryption contexts, plus the proven advantage that is typically negligible.

In particular, stream encryption with a keystream that is generated from the encryption context is perfectly secure in use cases where the encryption context is a nonce, but its security completely breaks down when re-using encryption contexts. Therefore, if we wish security in case of repeating encryption contexts, we must use a more elaborate encryption mechanism than stream encryption.

An example of protocol with bi-directional communications can be found in the full version [11].

2.4 Security of the jammin cipher in the OAE2 security model

We demonstrate the OAE2 security of the jammin cipher by proving an upper bound on the distinguishing advantage between the jammin cipher and OAE2 ideal-world system Rand2C. Concretely, referring to the OAE2c security definition [25, Fig. 6], we prove a tight bound for the case that the ciphertext expansion is t bits.

Theorem 1. *Let $\mathcal{J}^{+t}$ be the jammin cipher with* $\mathrm{WrapExpand}(p) = p+t$. *Then, for any adversary $\mathcal{D}$ that makes at most q queries, we have*

$$\mathbf{Adv}^{\text{oae2-priv}}_{\mathcal{J}^{+t}}(\mathcal{D}) \leq \frac{q}{2^{t+1}} \quad \textit{and} \quad \mathbf{Adv}^{\text{oae2-auth}}_{\mathcal{J}^{+t}}(\mathcal{D}) = 0\,.$$

Furthermore, when the encryption context is a nonce, we have

$$\mathbf{Adv}^{\text{oae2-priv}}_{\mathcal{J}^{+t}}(\mathcal{D}) = \mathbf{Adv}^{\text{oae2-auth}}_{\mathcal{J}^{+t}}(\mathcal{D}) = 0\,.$$

The proof can be found in the full version [11].

Our operational jammin cipher is hence fully indistinguishable from the non-operational Rand2C by a nonce-respecting adversary and defines the exact same security concept in that case. In case the encryption context is not a nonce, they can be distinguished only and exclusively by a property of Rand2C that makes it non-operational: non-injective encryption.

In [25, Proposition 2] the authors provide similar bounds for Ideal2B and obtain $\mathbf{Adv}^{\text{oae2-priv}}_{\text{Ideal2B}}(\mathcal{D}) \leq q^2/2^t$ and $\mathbf{Adv}^{\text{oae2-auth}}_{\text{Ideal2B}}(\mathcal{D}) \leq \ell/2^t$ with ℓ the number of messages in a single session. Thus, the jammin cipher is closer to the security definition Rand2C than Ideal2B is.

3 Deck functions

A *deck function* is a keyed function that takes a sequence of strings and returns a pseudorandom string of arbitrary length and that can be computed incrementally. Here *deck* stands for *Doubly-Extendable Cryptographic Keyed* function.

Definition 4 ([15]). *A* deck function F *takes as input a secret key $K \in \mathcal{K}_F$ and a sequence of an arbitrary number of strings $X^{(0)}; \ldots; X^{(m-1)} \in (\mathbb{Z}_2^*)^+$, produces a string of bits of arbitrary length and takes from it the range starting from a specified offset $q \in \mathbb{N}$ and for a specified length $n \in \mathbb{N}$. We denote this as*

$$Z = 0^n + F_K\left(X^{(0)}; \ldots; X^{(m-1)}\right) \ll q\ .$$

A deck function must allow efficient incremental computing, as described below, and typically comes with a security claim, see Section 3.1.

By efficient incremental computing we mean the following: by keeping state after computing an output for input $X = X^{(0)}; \ldots; X^{(m-1)}$, computing an output for $X; Y^{(0)}; \ldots; Y^{(n-1)}$ should have cost independent of X. In addition, by keeping state after computing $\mathtt{0}^n + F_K\left(X^{(0)}; \ldots; X^{(m-1)}\right) \ll q$, computing $\mathtt{0}^m + F_K\left(X^{(0)}; \ldots; X^{(m-1)}\right) \ll (q+n)$ should have cost independent of n or q.

Regarding the notation, we assume that the number of bits that the deck function outputs is determined by the context. For instance, in the expression $X + F_K(\ldots)$, we assume that the deck function outputs $|X|$ bits. Also, in $X + (F_K(\ldots)||Y)$, the deck function outputs $|X| - |Y|$ bits so that the string inside the brackets matches X in length.

3.1 Security claim

A deck function equipped with a fixed unknown random key should behave like a random oracle. We call this pseudorandom function (PRF) security.

Definition 5. *The advantage of an adversary $\mathcal{D}$ in distinguishing a deck function F from a random oracle $\mathcal{RO}$ is:*

$$\boldsymbol{Adv}_F^{\mathrm{prf}}(\mathcal{D}) = \left|\mathbb{P}\left[K \xleftarrow{\$} \mathcal{K}_F : \mathcal{D}^{F_K} = 1\right] - \mathbb{P}\left[\mathcal{D}^{\mathcal{RO}} = 1\right]\right| .$$

Here $\mathcal{RO}$ is a random oracle that takes as input a string sequence. We define the PRF advantage of a deck function $\boldsymbol{Adv}_F^{\mathrm{prf}}$ as

$$\boldsymbol{Adv}_F^{\mathrm{prf}}(\overline{R}) = \sup_{\mathcal{D} \in \mathcal{D}(\overline{R})} \boldsymbol{Adv}_F^{\mathrm{prf}}(\mathcal{D}) ,$$

with $\mathcal{D}(\overline{R})$ the set of all distinguishers with given resource limits $\overline{R}$. Here, we define the resource vector $\overline{R}$ in a rather abstract way, and in practice it typically comprises the data complexity M and the computational complexity N quantified in some well-defined unit.

Expressions for the PRF advantage of a particular deck function is not something that can be measured or proven. Rather, they are useful in security claims. For a particular deck function one can claim an upper bound on the PRF advantage and this serves as a challenge for cryptanalysts. For designers of cryptographic schemes making use of the deck function, they can serve as a security specification: Assuming the bound holds, it allows determining the security strength of the scheme. For the validity of the underlying assumption, one has no choice but to rely on cryptanalysis.

3.2 Multi-key security

In a multi-user setting with u users, we can adapt Definition 5 and replace the key K with a key array $\boldsymbol{K}$ drawn from $\mathcal{K}_F^u$. The adversary has to distinguish between u independently keyed deck functions and u independent random oracles:

$$\mathbf{Adv}_F^{\mathrm{prf}}(\mathcal{D}) = \left|\mathbb{P}\left[\boldsymbol{K} \xleftarrow{\$} \mathcal{K}_F^u : \mathcal{D}^{F_{K_1}, \ldots, F_{K_u}} = 1\right] - \mathbb{P}\left[\mathcal{D}^{\mathcal{RO}_1, \ldots, \mathcal{RO}_u} = 1\right]\right| .$$

The deck functions are independently keyed, that is, the keys are drawn from $\mathcal{K}_F$ with replacement. Consequently, $\mathbf{Adv}^{\mathrm{prf}}_F(\mathcal{D})$ cannot be smaller than the probability of collision within u draws in $\mathcal{K}_F$, or approximately $\frac{\binom{u}{2}}{|\mathcal{K}_F|}$ if u is small compared to the square root of the key space size.

Exhaustive key search is always possible, limiting the security of the deck function to $\log |\mathcal{K}_F|$ bits. If the adversary has access to the outputs of $F_{K_1}, \ldots, F_{K_u}$ for the same input, a single key guess has u chances of hitting one of the keys, leading to a security degradation of $\log u$ bits. Yet, one can avoid this degradation if the adversary is forced to feed different instances with different inputs, e.g., if the input is prefixed with a unique identifier. Whether adversary $\mathcal{D}$ has such a restriction is part of its resources $\overline{R}$.

3.3 Examples of deck functions

Deck functions can be built in many ways and two established constructions for building them from cryptographic permutations are the keyed duplex construction [17] and farfalle [8]. For the former, we can mention Strobe [23] and Xoodyak [14] as concrete instantiations. For the latter, Kravatte [8] and Xoofff [15] are two farfalle instantiations making use of the Keccak-f and Xoodoo permutations respectively. Another example of deck function is Subterranean-deck as part of the Subterranean 2.0 cipher suite [16].

A deck function can be built from other primitives and guarantee a certain PRF security level on the condition that the underlying primitive satisfies some security definition. For instance, we can imagine that a deck function can be fairly naturally built as a mode on top of a tweakable block cipher [29]. First, we compress the input through a secure MAC construction such as PMAC1 [37] or ZMAC [27], with slight adaptations for the multi-string input support. Then, we generate the output by processing the MAC through the tweakable block cipher, for instance with the tweak as a counter albeit in a different domain than during the compression. It is plausible that this construction can be proven PRF-secure assuming the tweakable block cipher to have tweakable PRP security.

3.4 Basic applications

Deck function can readily be used for stream encryption, authentication, and (nonce-based) AE of single messages.

One can use a deck function for stream encryption by taking as input a *diversifier* D and use the output to encrypt a plaintext P as $C \leftarrow P + F_K(D)$ and decrypt again as $P \leftarrow C + F_K(D)$. If the diversifier D is a nonce and F_K is random oracle, this is one-time pad encryption and so achieves perfect secrecy. Information leakage of this stream cipher is upper bounded by the PRF distinguishing advantage of the deck function. We refer to notion of indistinguishability from random bits under an adaptive chosen-plaintext-and-message-number attack, or IND$ [38]. This shows the following proposition:

Proposition 3. *Let $\mathcal{D}$ be any adversary attacking this stream cipher Π. Then there is an adversary $\mathcal{D}'$ using the same resources as $\mathcal{D}$ such that*

$$\mathbf{Adv}^{\mathrm{ind\$}}_{\Pi}(\mathcal{D}) \le \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{D}') \,.$$

One can use a deck function as a MAC function returning a t-bit tag by taking as input the *message* P and truncate the output to t: bits $T \leftarrow \mathtt{0}^t + F_K(P)$. One can verify a tag by taking as input the message P and its tag T and check whether $T + F_K(P)$ equals $\mathtt{0}^t$. If so, we say (P, T) verifies successfully. We speak of forgery if an adversary can find a (message,tag) pair (P, T), with T not generated in a tag generation query and that verifies successfully. Plugging in a random oracle for F_K would give a forgery success probability of $q/2^t$ with q the number of tag verification queries. It follows that the forgery success probability of our MAC function is at most by $q/2^t$ plus the PRF distinguishing advantage of the deck function. We hence prove the following proposition, see also [28, Section 4.4]:

Proposition 4. *Let $\mathcal{D}$ be any adversary attacking this authentication scheme Π. Then there exists an adversary $\mathcal{D}'$ using equivalent resources as $\mathcal{D}$ such that*

$$\mathbf{Adv}^{\text{uf-cma}}_{\Pi}(\mathcal{D}) \le \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{D}') + \frac{q_{\mathrm{ver}}}{2^t} \,,$$

with $\mathcal{D}$ making q_{ver} verification queries. The equivalence of resources means that the queries to the tag generation and tag verification methods are translated into queries to F of same length.

From this, AE with a deck function in an encrypt-then-MAC fashion is immediate. The plaintext is encrypted as $Z \leftarrow P + F_K(A)$, with A associated data that should be a nonce (it may contain a diversifier). Then a tag is computed as $T \leftarrow \mathtt{0}^t + F_K(A; Z)$. The cryptogram (Z, T) can be first verified and then decrypted if the tag is correct. Apart from string encoding details, this is a non-session special case of Deck-PLAIN, covered in the next section.

4 Deck-PLAIN

We specify in Algorithm 2 a deck function mode for nonce-based session-supporting AE called Deck-PLAIN. It allows two parties to exchange a sequence of messages, each consisting of associated data and plaintext. At sending end it wraps a message by encrypting the plaintext to a ciphertext and appending a tag that authenticates the sequence of all messages up to that point. At receiving end it unwraps a cryptogram by verifying the tag and, if correct, it decrypts the ciphertext; otherwise, it will return an error.

Deck-PLAIN offers the same interface as the jammin cipher. The only difference is upon initialization, where the jammin cipher takes an identifier as input, while Deck-PLAIN takes a secret key, in particular from an array of keys to be able to model multi-key support. It has two length parameters: the tag length t that determines the security level and and an alignment unit length ℓ that is related to an implementation optimization as detailed below.

In the individual messages both associated data and plaintext are optional. We call messages without plaintext *authentication-only* messages and messages without associated data *plaintext-only* messages. Deck-PLAIN even supports empty messages for the purpose of authenticated acknowledgments.

If a key is used more than once, the associated data of the first message of the session **must** be a nonce per key, e.g., a session counter. One may choose to have an authentication-only first message. The corresponding tag is then called a *startup tag*. Verification of a startup tag allows the receiver of the message to authenticate the origin of the session start request including the session counter.

4.1 Inner workings

Similar to the jammin cipher, Deck-PLAIN accumulates the sequence of messages in a data element called history. Concretely, this is the sequence of associated data and plaintexts of messages received and differs only from history in the jammin cipher by the explicit encoding used.

In a wrap call, Deck-PLAIN encrypts a plaintext by adding to it a keystream that is the output of the underlying deck function with input the *context*. This context is the history followed by A of the message. Clearly, the encryption context is the same as in the jammin cipher. Initialization of a session loads the key in the deck function and initializes the history to an empty sequence.

Deck-PLAIN performs the wrapping of a message in two steps:

1. **Encryption:** It extracts keystream from the deck function and adds it to the plaintext, yielding the ciphertext.
2. **Tag generation:** It appends associated data and ciphertext to the history and extracts the tag from the deck function.

Unwrapping is similar. Tag verification is performed before decryption.

In consecutive plaintext-only wrap or unwrap calls, Deck-PLAIN reserves the first t bits of deck function outputs for tags and the remaining ones for keystream. It takes keystream from an offset that is the smallest multiple of ℓ not shorter than t. So Deck-PLAIN requires only one deck function call per message in this important use case.

For authentication-only messages Deck-PLAIN skips the en(de)cryption step and the absorbing of ciphertext. For plaintext-only messages it skips the absorbing of associated data, except for a blank message where it absorbs the empty associated data. To make the mapping from sequences of messages to the history injective, Deck-PLAIN appends frame bits to associated data and ciphertext strings for domain separation before appending to the history. In particular, ciphertext strings end with `1` and associated data strings with `00` (in an authentication-only message) or `10` (otherwise).

4.2 Security analysis

To be secure, Deck-PLAIN relies on the encryption context to be a nonce, as it otherwise leaks the difference between two plaintexts, as for stream ciphers. If

Algorithm 2 Definition of Deck-PLAIN(F, t, ℓ)

Parameters: deck function F, tag length $t \in \mathbb{N}$ and alignment unit length $\ell \in \mathbb{N}$
Let offset $= \ell \left\lceil \frac{t}{\ell} \right\rceil$: the smallest multiple of ℓ not smaller than t

Instance constructor: $\mathsf{init}(\boldsymbol{K}, i)$ taking key array $\boldsymbol{K}$, key index i
$(\mathsf{inst}.K, \mathsf{inst}.\mathrm{history}) \leftarrow (\boldsymbol{K}[i], \varnothing)$
return Deck-PLAIN instance
Note: in the sequel, K, history denote the attributes of inst

Instance cloner: $\mathsf{inst}.\mathsf{clone}()$
return new instance inst' with all attributes (K, history) copied from inst

Interface: $\mathsf{inst}.\mathsf{wrap}(A, P)$ returns C
if $|P| = 0$ **then**
  history $\leftarrow$ history; $A||\mathtt{00}$
else if $|A| > 0$ or history $= \varnothing$ **then**
  $\mathsf{context} \leftarrow$ history; $A||\mathtt{10}$
  $Z \leftarrow P + F_K(\mathsf{context})$
  history $\leftarrow \mathsf{context}; Z||\mathtt{1}$
else
  $\mathsf{context} \leftarrow$ history
  $Z \leftarrow P + F_K(\mathsf{context}) \ll$ offset
  history $\leftarrow \mathsf{context}; Z||\mathtt{1}$
$T \leftarrow \mathtt{0}^t + F_K(\mathrm{history})$
return $C = Z||T$

Interface: $\mathsf{inst}.\mathsf{unwrap}(A, C)$ returns P or $\perp$
if $|C| < t$ **then return** $\perp$
Parse C in Z and T
if $|Z| = 0$ **then**
  history$'$ $\leftarrow$ history; $A||\mathtt{00}$
else if $|A| > 0$ or history $= \varnothing$ **then**
  history$'$ $\leftarrow$ history; $A||\mathtt{10}; Z||\mathtt{1}$
else
  history$'$ $\leftarrow$ history; $Z||\mathtt{1}$
$T' \leftarrow \mathtt{0}^t + F_K(\mathrm{history}')$
if $T' \neq T$ **then return** $\perp$
if $|A| > 0$ or history $= \varnothing$ **then**
  $\mathsf{context} \leftarrow$ history; $A||\mathtt{10}$
  $P \leftarrow Z + F_K(\mathsf{context})$
else
  $\mathsf{context} \leftarrow$ history
  $P \leftarrow Z + F_K(\mathsf{context}) \ll$ offset
history $\leftarrow$ history$'$
return P

the encryption context is a nonce, Deck-PLAIN can be distinguished from the jammin cipher only by a forgery or by distinguishing the deck function from a random function, as captured in the following theorem. Multi-key security is covered by the $\mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{D}')$ term, see Section 3.2. In particular, one can avoid multi-key security degradation by ensuring that the encryption context is different per instance of Deck-PLAIN.

Theorem 2. *Let $\mathcal{D}$ be any fixed deterministic adversary whose goal is to distinguish* Deck-PLAIN(F, t, ℓ) *from $\mathcal{J}^{+t}$, the jammin cipher with* WrapExpand$(p) = p + t$. *If in the queries of $\mathcal{D}$ the encryption context is a nonce, there exists an adversary $\mathcal{D}'$ using the same resources as $\mathcal{D}$ such that*

$$\Delta_{\mathcal{D}}(\text{Deck-PLAIN}(F, t, \ell)\ ;\ \mathcal{J}^{+t}) \leq \frac{q_{\mathsf{unwrap}}}{2^t} + \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{D}'),$$

with q_{unwrap} the number of unwrap calls $\mathcal{D}$ makes.

We now introduce the proof technique and given the proof in Section 4.4.

4.3 The H-coefficient technique

Our proofs use the H-coefficient technique from Patarin [35]. We will follow the adaptation of Chen and Steinberger [12]. Consider any information-theoretic deterministic adversary $\mathcal{A}$ whose goal is to distinguish $\mathcal{O}$ from $\mathcal{P}$, with its advantage denoted $\Delta_{\mathcal{A}}(\mathcal{O}\ ;\ \mathcal{P})$. The interaction of $\mathcal{A}$ with its oracles, either $\mathcal{O}$ or $\mathcal{P}$, will be recorded in a *transcript* τ. Denote by $D_{\mathcal{O}}$ (resp. $D_{\mathcal{P}}$) the probability distribution of transcripts that can be obtained from interaction with $\mathcal{O}$ (resp. $\mathcal{P}$). Call a transcript τ *attainable* if $\Pr(D_{\mathcal{P}} = \tau) > 0$. Denote by $\mathcal{T}$ the set of attainable transcripts, and consider any partition $\mathcal{T} = \mathcal{T}_{\mathrm{good}} \cup \mathcal{T}_{\mathrm{bad}}$ into "good" and "bad" transcripts. The H-coefficient technique states the following [12].

Lemma 1 (H-coefficient Technique). *Consider a fixed information-theoretic deterministic adversary $\mathcal{A}$ whose goal is to distinguish $\mathcal{O}$ from $\mathcal{P}$. Let ε be such that for all $\tau \in \mathcal{T}_{\mathrm{good}}$:* $\Pr(D_{\mathcal{O}} = \tau) / \Pr(D_{\mathcal{P}} = \tau) \geq 1 - \varepsilon$. *Then,* $\Delta_{\mathcal{A}}(\mathcal{O}\ ;\ \mathcal{P}) \leq \varepsilon + \Pr(D_{\mathcal{P}} \in \mathcal{T}_{\mathrm{bad}})$.

The H-coefficient technique can thus be used to bound a distinguishing advantage in the terminology of Definition 1. In our proofs below, we use the special case where $\Pr(D_{\mathcal{O}} = \tau) \geq \Pr(D_{\mathcal{P}} = \tau)$ for all $\tau \in \mathcal{T}_{\mathrm{good}}$, so that $\Delta_{\mathcal{A}}(\mathcal{O}\ ;\ \mathcal{P}) \leq \Pr(D_{\mathcal{P}} \in \mathcal{T}_{\mathrm{bad}})$, and we set $\mathcal{O}$ to the jammin cipher and $\mathcal{P}$ to the real world.

4.4 Proof of Theorem 2

Proof. We use a hybrid argument and replace the deck function with a random oracle before comparing Deck-PLAIN with the jammin cipher, i.e.,

$$\begin{aligned}\Delta_{\mathcal{D}}(\text{Deck-PLAIN}(F, t, \ell)\ ;\ \mathcal{J}^{+t}) \\ \leq \Delta_{\mathcal{D}''}(\text{Deck-PLAIN}(\mathcal{RO}, t, \ell)\ ;\ \mathcal{J}^{+t}) + \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{D}'),\end{aligned}$$

where $\mathcal{D}''$ has the same resources as $\mathcal{D}$. Here Deck-PLAIN$(\mathcal{RO}, t, \ell)$ is a slight abuse of notation. It means that the instance constructor chooses the i-th random oracle from an array and that F_K refers to $\mathcal{RO}_i$ in Algorithm 2.

We then use Lemma 1 with $\mathcal{O} = \mathcal{J}^{+t} \triangleq \mathcal{J}$ and $\mathcal{P} =$ Deck-PLAIN$(\mathcal{RO}, t, \ell)$. In this proof, we use the session syntax of the jammin cipher. This is w.l.o.g. as in Deck-PLAIN the history is an encoding of the more abstract history in $\mathcal{J}$.

We define a transcript τ as a sequence of records of the form

$$(\mathsf{wrap/unwrap}, \mathsf{context}, P, C),$$

where the first component indicates the type of call made and the context is the combination of the history as in the definition of $\mathcal{J}$ and A of the wrap/unwrap call. In a wrap record, P is a parameter and C is the returned value, with $C \neq \perp$. In an unwrap record, C is a parameter and P is a return value and may contain an error code $\perp$. We ignore in the transcript wrap records with equal tuple $(\mathsf{context}, P)$ and unwrap records with equal tuple $(\mathsf{context}, C)$. This is w.l.o.g. as both worlds act deterministically. Similarly, we ignore in the transcript unwrap records that have the same tuple $(\mathsf{context}, P, C)$ as a wrap record. This is w.l.o.g. as both worlds behave consistently in this respect. This yields a simple definition of forgery, namely the presence of a successful unwrap record in the transcript.

We have one type of bad event: a successful forgery. $\mathcal{T}_{\text{bad}}$ is the set of transcripts containing a record $(\mathsf{unwrap}, \mathsf{context}, P, C)$ with $P \neq \perp$. In a forgery attempt, unwrap compares a tag to a tag generated with the underlying $\mathcal{RO}$ applied to a unique input. As the latter is a uniformly generated t-bit string, the probability that they are equal is 2^{-t}, hence $\Pr(D_{\mathcal{P}} \in \mathcal{T}_{\text{bad}}) \leq \frac{q_{\mathsf{unwrap}}}{2^t}$ after q_{unwrap} calls to unwrap.

We now prove that, for all $\tau \in \mathcal{T}_{\text{good}}$, we have $\Pr(D_{\mathcal{J}} = \tau) \geq \Pr(D_{\mathcal{P}} = \tau)$, hence $\varepsilon = 0$ in Lemma 1. In both worlds, the cryptogram bits are generated randomly and independently for different contexts, so we can partition the transcript records per context and take the probability as the product of the probabilities over the different contexts. We will now consider a subset of the transcript for a given context value.

As the context is unique per wrap call for non-empty plaintexts, there can be only one of the form $(\mathsf{wrap}, \mathsf{context}, P \neq \epsilon, C)$ and one of the form $(\mathsf{wrap}, \mathsf{context}, \epsilon, C_\epsilon \neq C)$.

Upon an unsuccessful unwrap query, the jammin cipher returns $\perp$ as it avoids forgeries and hence contributes a factor 1 to the probability. Upon a wrap query, the jammin cipher selects C from a set of cardinality at most $2^{|P|+t}$ and hence contributes a factor at least $2^{-(|P|+t)}$ to $\Pr(D_{\mathcal{J}} = \tau)$. It may return an error, but thanks to Proposition 2, this would require $q_{\mathsf{unwrap}} \geq 2^t$.

Upon an unsuccessful unwrap query, $\mathcal{P} =$ Deck-PLAIN$(\mathcal{RO}, t, \ell)$ returns $\perp$ in a good transcript and this contributes at most 1 to $\Pr(D_{\mathcal{P}} = \tau)$. Upon a wrap query, $\mathcal{P}$ computes the value $C = Z||T$ with $Z = P + \mathcal{RO}(\mathsf{context})$, context = history; A (or $P + \mathcal{RO}(\text{history}) \ll$ offset when $A = \epsilon$ and history $\neq \varnothing$) and $T = \mathcal{RO}(\text{updated history})$. Thanks to the fact that upon wrap the context is unique and $\mathcal{P}$ takes tags and keystream in different domains or from different parts of

the RO output stream, it contributes a factor exactly $2^{-(|P|+t)}$ to $\Pr(D_{\mathcal{P}} = \tau)$. A wrap record with $P = \epsilon$ contributes a factor 2^{-t} to $\Pr(D_{\mathcal{P}} = \tau)$.

This shows that $\Pr(D_{\mathcal{J}} = \tau) \geq \Pr(D_{\mathcal{P}} = \tau)$ and concludes the proof. □

5 Feistel network modes

The security of Deck-PLAIN breaks down when the encryption context is not a nonce. In this section, we introduce four different modes of deck functions that are more robust against nonce misuse. Two of the modes make optimal use of the redundancy: for t-bit security they only require a plaintext expansion by t bits. Moreover, two of them provide protection against the accidental release of unverified decrypted ciphertext (a.k.a. release of unverified plaintext or RUP [3]).

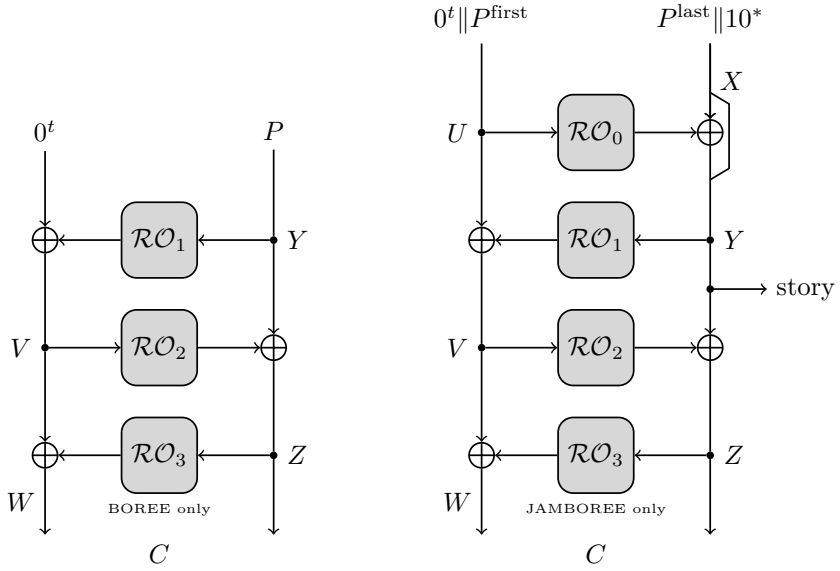

Fig. 1: Feistel network inside the different modes, Deck-BO(REE) on the left and Deck-JAMBO(REE) on the right.

It turns out that different modes like Synthetic Initial Value (SIV) [40], Robust IV (RIV) [2] and wide-block ciphers [30,31] can all be expressed under the hood of a Feistel network. We here give an intuitive overview of these constructions, starting with the simplest case: SIV. Consider Figure 1 (left) with only the first two rounds. The left branch is initialized with t bits set to zero, while the right branch contains the plaintext. After the first round, V is a pseudorandom function of the plaintext and becomes the tag. We use V also as a synthetic diversifier in the next round, and encrypt the plaintext $Y = P$ by adding to it a keystream that depends on V.

In case the implementation (accidentally) releases unverified decrypted ciphertexts, an adversary can obtain such for chosen values of V.

After querying unwrap with $C_0 = V||Z_0$ and $C_1 = V||Z_1$ and get unverified decrypted ciphertexts P_0 and P_1, she observes that $Z_0 + Z_1 = P_0 + P_1$. The RIV mode avoids this by adding a third round. The ciphertext Z serves as input to a third pseudorandom function to mask V. Compared to SIV, the adversary cannot control V at decryption anymore since she has access to W only.

To avoid collisions in V, SIV and RIV need to have t large enough. In case of unbounded nonce misuse, due to the birthday paradox we must take $t = 2s$ for s bits of security. Consider now Figure 1 (right). Compared to SIV and RIV, it adds a round at the beginning and the plaintext is spread onto the two branches, with t bits of redundancy on the left branch. This round compresses P into Y, and then we proceed as with SIV and RIV. The left and right branch must be wide enough to avoid collisions in Y, but this is decoupled from the expansion length t and we can now have $t = s$ for s bits of security. If a mode performs the first three rounds but not the last one, we obtain a variant of an SIV mode with optimal redundancy but no resistance to RUP.

We call our modes Deck-BO, Deck-BOREE, Deck-JAMBO and Deck-JAMBOREE and they make use of the Feistel network-based block cipher in Algorithm 3. This algorithm is parameterized with the deck function F and whether the optional first (jam) and last (ree) rounds are performed. A call to the block cipher takes as input a secret key K, a context (tweak) and the input already split into four parts $L_0||L_+||R_0||R_+$. The left branch is $L_0||L_+$ and the right branch is $R_0||R_+$. The first (resp. last) round affects only R_0 (resp. L_0). Additionally, the block cipher returns a history that is the combination of its context and the intermediate value Y. In Deck-BO(REE), Y coincides with the plaintext, while in Deck-JAMBO(REE) it is the compressed plaintext or *plaintext representative*. In all cases, Y needs to be absorbed when evaluating the block cipher and this allow the returned history not to have to be absorbed again, thanks to the incrementality of the deck function.

5.1 Deck-BO

Deck-BO, defined in Algorithm 4, combines the SIV approach [40] with the session support of Deck-PLAIN. Deck-BO wraps a message in three phases:

1. Tag generation: It generates the tag by applying the deck function to the context (history and A) and the plaintext of the message, if non-empty.
2. Encryption: If the plaintext is non-empty, it generates the ciphertext by adding to the plaintext the output of the deck function applied to the context extended with the tag.
3. It updates the history.

Unwrapping is similar. Deck-BO has a single length parameter: the tag length t. It applies domain separation between associated data and plaintext strings in the history, as well as between the generation of keystream and of tag.

Algorithm 3 Definition of block cipher B and its inverse.

Parameters: deck function F and round flags $\subseteq \{\mathsf{jam}, \mathsf{ree}\}$
Note: in the sequel, L is a shortcut notation for $L_0||L_+$ and R for $R_0||R_+$.

Interface: $O = B_{F,\text{flags}}(K, \mathsf{context}, L_0, L_+, R_0, R_+)$
if jam $\in$ flags **then**
  $R_0 \leftarrow R_0 + F_K\,(\mathsf{context}; L||\mathtt{001})$
$L \leftarrow L + F_K\,(\mathsf{context}; R||\mathtt{011})$
history $\leftarrow \mathsf{context}; R||\mathtt{011}$
$R \leftarrow R + F_K\,(\mathsf{context}; L||\mathtt{101})$
if ree $\in$ flags **then**
  $L_0 \leftarrow L_0 + F_K\,(\mathsf{context}; R||\mathtt{111})$
return (history, $L||R$)

Interface: $O = B^{-1}_{F,\text{flags}}(K, \mathsf{context}, L_0, L_+, R_0, R_+)$
if ree $\in$ flags **then**
  $L_0 \leftarrow L_0 + F_K\,(\mathsf{context}; R||\mathtt{111})$
$R \leftarrow R + F_K\,(\mathsf{context}; L||\mathtt{101})$
$L \leftarrow L + F_K\,(\mathsf{context}; R||\mathtt{011})$
history $\leftarrow \mathsf{context}; R||\mathtt{011}$
if jam $\in$ flags **then**
  $R_0 \leftarrow R_0 + F_K\,(\mathsf{context}; L||\mathtt{001})$
return (history, $L||R$)

In contrast to Deck-PLAIN, leakage of Deck-BO is limited to revealing plaintext equality under equal encryption context. To achieve that, Deck-BO computes the tag over the history with associated data and plaintext attached and then generates the keystream from the encryption context with this tag appended to it. Unless we have colliding tags for equal encryption contexts, keystreams are independent. Therefore, for its security Deck-BO relies on the absence of (rare) tag collisions. The security of Deck-BO is captured in Theorem 3.

Theorem 3. *Let $\mathcal{D}$ be any fixed deterministic adversary whose goal is to distinguish* Deck-BO(F,t) *from $\mathcal{J}^{+t}$, the jammin cipher with* WrapExpand$(p) = p+t$. *Then there exists an adversary $\mathcal{D}'$ using the same resources as $\mathcal{D}$ such that*

$$\Delta_{\mathcal{D}}(\text{Deck-BO}(F,t)\ ;\ \mathcal{J}^{+t}) \leq \frac{q_{\mathsf{unwrap}}}{2^t} + \sum_{\mathsf{context}} \frac{\binom{\sigma(\mathsf{context})}{2}}{2^t} + \mathbf{Adv}^{\text{prf}}_F(\mathcal{D}'),$$

with q_{unwrap} the number of unwrap calls that $\mathcal{D}$ makes and $\sigma(\mathsf{context})$ the number of wrap queries with $P \neq \epsilon$ for a given context *value.*

The second term is due to tags colliding for equal encryption contexts and it determines the length of the tag to achieve a certain security strength s. If the encryption context is a nonce, the term vanishes and it is sufficient to take $t = s$. In case of unbounded nonce misuse, it may reach $\frac{q^2_{\mathsf{wrap}}}{2^{t+1}}$ and we have to

Algorithm 4 Definition of Deck-BO(F, t) and Deck-BOREE(F, t)

Parameters: deck function F and expansion length t
$B = B_{F,\varnothing}$ for Deck-BO or $B = B_{F,\{\mathsf{ree}\}}$ for Deck-BOREE

Constructor: $\mathsf{init}(\boldsymbol{K}, i)$ taking key array $\boldsymbol{K}$, key index i
$(K, \text{history}) \leftarrow (\boldsymbol{K}[i], \varnothing)$
return instance

Interface: $\mathsf{wrap}(A, P)$ returning C
if $|P| = 0$ **then**
  history $\leftarrow$ history; $A||\mathtt{00}$
  return $C \leftarrow \mathtt{0}^t + F_K$ (history)
if $|A| = 0$ **then** context $\leftarrow$ history **else** context $\leftarrow$ history; $A||\mathtt{10}$
$(\text{history}, C) \leftarrow B(K, \mathsf{context}, \mathtt{0}^t, \epsilon, P, \epsilon)$
return C

Interface: $\mathsf{unwrap}(A, C)$ returning P or $\perp$
if $|C| = t$ **then**
  history$'$ $\leftarrow$ history; $A||\mathtt{00}$
  $P \leftarrow \epsilon$
  $C' \leftarrow \mathtt{0}^t + F_K$ (history$'$)
  if $C' \neq C$ **then return** $\perp$
else if $|C| > t$ **then**
  if $|A| = 0$ **then** context $\leftarrow$ history **else** context $\leftarrow$ history; $A||\mathtt{10}$
  $T||Z \leftarrow C$ such that $|T| = t$
  $(\text{history}', P') \leftarrow B^{-1}(K, \mathsf{context}, T, \epsilon, Z, \epsilon)$
  $L||P \leftarrow P'$ such that $|L| = t$
  if $L \neq \mathtt{0}^t$ **then return** $\perp$
else return $\perp$
history $\leftarrow$ history$'$
return P

set $t \geq 2s - 1$. In use cases where the number of times an encryption context is repeated can be upper bounded by 2^x, we can relax this to $t = s + x - 1$.

The proof can be found in the full version [11].

5.2 Deck-BOREE and release of unverified decrypted ciphertexts

Deck-BO does not tolerate the release of unverified decrypted ciphertexts when unwrapping. This leads to a distinguisher as detailed earlier. We introduce Deck-BOREE to address use cases where this is a concern. Deck-BOREE hides the tag value from the adversary by encrypting it using keystream computed from the ciphertext. The distinguisher described above for Deck-BO no longer works as the tag (SIV) depends on the ciphertext and decryption leads to independent keystreams and therefore independent decrypted ciphertexts. We define Deck-BOREE in Algorithm 4.

Theorem 4 formalizes the security of Deck-BOREE. For the release of unverified decrypted ciphertexts, we use an approach similar to indifferentiability [32]. In the real world, we extend the interface of the adversary with the value of the right branch (Y) after processing the unwrap query, as this is where the plaintext appears before the tag is verified. For the ideal world, such a right branch does not exist and we *simulate* it with independently distributed random bits, so without connection to any actual plaintexts. Infeasibility to distinguish the two systems with this extended interface implies that security is preserved even when releasing unverified decrypted ciphertext.

In addition, we grant the adversary the choice per query whether she gets the value of the right branch (or its simulated value). If not, she just receives $\perp$. So Theorem 4 also covers the case where the unverified decrypted ciphertexts are not disclosed, or only a limited number of them.

Theorem 4. *Let $\mathcal{D}$ be any fixed deterministic adversary whose goal is to distinguish* Deck-BOREE(F,t) *from $\mathcal{J}^{+t}$, the jammin cipher with* WrapExpand$(p) = p + t$. *In addition, this adversary has access to the unverified decrypted ciphertexts in the case of Deck-BOREE and to a random string of bits $|C| - t$ bits in the case of the jammin cipher. Then there exists an adversary $\mathcal{D}'$ using the same resources as $\mathcal{D}$ such that*

$$\Delta^{\mathrm{RUP}}_{\mathcal{D}}(\text{Deck-BOREE}(F,t)\ ;\ \mathcal{J}^{+t}) \leq \frac{q_{\mathsf{unwrap}}}{2^t} + \sum_{\mathsf{context}} \frac{\binom{\sigma'(\mathsf{context})}{2}}{2^t} + \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{D}'),$$

with q_{unwrap} the number of unwrap calls that $\mathcal{D}$ makes and $\sigma'(\mathsf{context})$ the number of wrap (resp. unwrap) queries with $P \neq \epsilon$ (resp. $|C| > t$ and the adversary accesses the unverified decrypted ciphertext) for a given context *value.*

The second term is due to (hidden) tag collisions for wrap call and unwrap calls with leakage for given encryption contexts. As for Deck-BO, it determines the length of the tag to achieve a certain security strength s and the same trade-offs apply. If the adversary does not access unverified decrypted ciphertext, unwrap queries do not contribute to $\sigma'(\mathsf{context})$ and we get the same bound as in Theorem 3 for Deck-BO.

The proof can be found in the full version [11].

5.3 Deck-JAMBO and optimal redundancy

Deck-JAMBO is an enhancement of Deck-BO in that it resulting in less required expansion at the cost of an additional round at the beginning in order to protect against chosen plaintext attacks. With Deck-JAMBO, it is possible to take advantage of redundancy that is already present in the plaintext, as long as it resides in the left branch of the Feistel network. We define it in Algorithm 5.

We leave the specifications of how to split the input of the block cipher into left and right parts out of the definition of Deck-JAMBO and Deck-JAMBOREE. The reason is that the most efficient way to do so may vary with the particular

deck function in use. For instance, for farfalle-based deck functions, one may wish the left part of the input to fit in exactly one block after padding. Such specific technicalities do not belong in the definition of a general-purpose mode.

The split cuts the expanded plaintext or cryptogram into four parts. We formalize this with three functions that must satisfy some properties: plaintext expansion and extraction and a split function.

First, the expand function takes as input the plaintext P and the expansion length t and returns the expanded plaintext $P' = \text{expand}(P, t)$ of the form $0^t||P||10^*$. The number of zero bits at the end may depend on the length of P but shall not depend on its value. This function must ensure that $|P'| \geq 4t$. The expand function implicitly defines a WrapExpand function, namely,

$$\text{WrapExpand}(|P|) = |\text{expand}(P, t)| \ .$$

For $P = \epsilon$, Deck-JAMBO has a special treatment and the resulting cryptogram has $|C| = t$ bits. So, we can set $\text{WrapExpand}(0) = t$ and therefore the implicitly defined WrapExpand function is t-expanding by construction.

Second, we define a plaintext extraction function called $\text{extract}(P', t)$ that returns $\perp$ if P' does not start with 0^t or cannot be unpadded, and extracts P otherwise. Naturally, we require that $\text{extract}(\text{expand}(P, t)) = P$ for any P. Note that the behavior of this function is fixed and cannot be customized.

Third, the split function takes as input the expanded plaintext P' or ciphertext C and the expansion length t, and it returns a tuple $(L_0, L_+, R_0, R_+) = \text{split}(\alpha, t)$ such that $\alpha = L_0||L_+||R_0||R_+$, $|L_0| \geq 2t$ and $|R_0| \geq 2t$. Here again, the lengths of the four parts may depend on the length of the input string but not on its value. If the input string is shorter than $4t$ bits, it returns an error.

Compared to Deck-BO, we renamed the history to *story* as it is no longer guaranteed that the mapping of the sequence of messages to this sequence of strings is injective. In particular, we do not append plaintexts but rather plaintext representatives. Different plaintexts with colliding plaintext representatives are rare, and we treat them as bad events in the proof.

The security of Deck-JAMBO is captured in the theorem below. Compared to Deck-BO, the expansion parameter t can be equal to the security strength s in all cases. Collisions that happen on the left or right branch are bad events, but as the branches are at least $2t$ bits wide, these are rare.

Theorem 5. *Let $\mathcal{D}$ be any fixed deterministic adversary whose goal is to distinguish* Deck-JAMBO$(F, t, \text{expand}, \text{split})$ *from $\mathcal{J}^{t,\text{expand}}$, the jammin cipher with* WrapExpand *that follows from t and the chosen expand function (or $\mathcal{J}$ for short). Then there is an adversary $\mathcal{D}'$ using the same resources as $\mathcal{D}$ such that*

$$\Delta_{\mathcal{D}}(\text{Deck-JAMBO}(F, \dots) \ ; \ \mathcal{J}) \leq \frac{q_{\mathsf{unwrap}}}{2^t} + \sum_{\mathsf{context}} \frac{\binom{\sigma(\mathsf{context})}{2}}{2^{2t-1}} + \mathbf{Adv}_F^{\text{prf}}(\mathcal{D}'),$$

with q_{unwrap} the number of unwrap calls that $\mathcal{D}$ makes and $\sigma(\mathsf{context})$ the number of wrap queries with $P \neq \epsilon$ for a given context *value.*

The proof can be found in the full version [11].

Algorithm 5 Definition of Deck-JAMBO(REE)$(F, t, \text{expand}, \text{split})$

Parameters: deck function F, expansion length t, expand and split functions $B = B_{F,\{\mathsf{jam}\}}$ for Deck-JAMBO or $B = B_{F,\{\mathsf{jam},\mathsf{ree}\}}$ for Deck-JAMBOREE

Constructor: $\mathsf{init}(\boldsymbol{K}, i)$ taking key array $\boldsymbol{K}$, key index i
$(K, \text{story}) \leftarrow (\boldsymbol{K}[i], \varnothing)$
return instance

Interface: $\mathsf{wrap}(A, P)$ returning C
if $|P| = 0$ **then**
  story $\leftarrow$ story; $A||\mathsf{00}$
  return $C \leftarrow \mathsf{0}^t + F_K(\text{story})$
if $|A| = 0$ **then** context $\leftarrow$ story **else** context $\leftarrow$ story; $A||\mathsf{10}$
$P' \leftarrow \text{expand}(P, t)$
$(L_0, L_+, R_0, R_+) \leftarrow \text{split}(P', t)$
$(\text{story}, C) \leftarrow B(K, \mathsf{context}, L_0, L_+, R_0, R_+)$
return C

Interface: $\mathsf{unwrap}(A, C)$ returning P or $\perp$
$\text{story}' \leftarrow$ story
if $|C| = t$ **then**
  $\text{story}' \leftarrow \text{story}'; A||\mathsf{00}$
  $C' \leftarrow \mathsf{0}^t + F_K(\text{story}')$
  if $C' = C$ **then** $P \leftarrow \epsilon$ **else** $P \leftarrow \perp$
else if $\text{split}(C, t) \neq \perp$ **then**
  if $|A| = 0$ **then** context $\leftarrow$ story **else** context $\leftarrow$ story; $A||\mathsf{10}$
  $(L_0, L_+, R_0, R_+) \leftarrow \text{split}(C, t)$
  $(\text{story}', P') \leftarrow B^{-1}(K, \mathsf{context}, L_0, L_+, R_0, R_+)$
  $P \leftarrow \text{extract}(P', t)$
else $P \leftarrow \perp$
if $P \neq \perp$ **then** story $\leftarrow \text{story}'$
return P

5.4 Deck-JAMBOREE

Deck-JAMBOREE combines the advantages of Deck-BOREE and Deck-JAMBO in a natural way. For encryption it makes use of a wide tweakable block cipher such as AEZ [24] but rather specified in terms of a deck function, like Double-decker [22]. For authentication, it relies on the redundancy in the expanded plaintext presented to this block cipher.

The security of Deck-JAMBOREE is captured in the theorem below. Like Deck-JAMBO, the expansion parameter t can be equal to the security strength s. And like Deck-BOREE, it is secure even in the case of the release of unverified decrypted ciphertext. The RUP model is defined similarly, with the difference that there is no clear split anymore between the ciphertext and the tag as in Deck-BOREE. Hence, the adversary has access to the entire unverified decrypted cryptogram, which would contain the expanded plaintext in a successful unwrap.

Theorem 6. *Let $\mathcal{D}$ be any fixed deterministic adversary whose goal is to distinguish* Deck-JAMBOREE$(F, t, \text{expand}, \text{split})$ *from $\mathcal{J}^{t,\text{expand}}$, the jammin cipher with* WrapExpand *that follows from t and the chosen expand function (or $\mathcal{J}$ for short). In addition, this adversary has access to the unverified decrypted cryptograms in the case of Deck-JAMBOREE and to a random string of bits $|C|$ bits in the case of the jammin cipher. Then there exists an adversary $\mathcal{D}'$ using the same resources as $\mathcal{D}$ such that*

$$\Delta^{\text{RUP}}_{\mathcal{D}}(\text{Deck-JAMBOREE}(F, \dots)\ ;\ \mathcal{J}) \leq \frac{q_{\mathsf{unwrap}}}{2^t} + \sum_{\mathsf{context}} \frac{\binom{\sigma'(\mathsf{context})}{2}}{2^{2t-1}} + \mathbf{Adv}^{\text{prf}}_{F}(\mathcal{D}'),$$

with q_{unwrap} the number of unwrap calls that $\mathcal{D}$ makes and $\sigma'(\mathsf{context})$ the number of wrap (resp. unwrap) queries with $P \neq \epsilon$ (resp. $|C| > t$ and the adversary accesses the unverified decrypted cryptogram) for a given context *value.*

The proof can be found in the full version [11].

6 Conclusions

We found that proving the security of the deck function-based modes is relatively easy and gives strong bounds that are tight, as the bounds account only for simple bad events like tag guessing and internal collisions. New modes are relatively easy to design, and this opens the door to more tailored schemes for niche applications, but we leave this as future work.

Acknowledgements

Joan Daemen is supported by the European Research Council under the ERC advanced grant agreement under grant ERC-2017-ADG Nr. 788980 ESCADA.

References

1. Abed, F., Fluhrer, S., Forler, C., List, E., Lucks, S., McGrew, D., Wenzel, J.: The POET family of on-line authenticated encryption schemes. CAESAR Submission (2014)
2. Abed, F., Forler, C., List, E., Lucks, S., Wenzel, J.: RIV for robust authenticated encryption. In: Peyrin, T. (ed.) Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9783, pp. 23–42. Springer (2016)
3. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: How to securely release unverified plaintext in authenticated encryption. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology - ASIACRYPT 2014, Proceedings, Part I. Lecture Notes in Computer Science, vol. 8873, pp. 105–125. Springer (2014)
4. Barbosa, M., Farshim, P.: Indifferentiable authenticated encryption. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10991, pp. 187–220. Springer (2018)

5. Bellare, M., Boldyreva, A., Knudsen, L.R., Namprempre, C.: Online ciphers and the hash-cbc construction. In: Kilian, J. (ed.) Advances in Cryptology - CRYPTO 2001, Proceedings. Lecture Notes in Computer Science, vol. 2139, pp. 292–309. Springer (2001)
6. Bellare, M., Kohno, T., Namprempre, C.: Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the encode-then-encrypt-and-mac paradigm. ACM Trans. Inf. Syst. Secur. 7(2), 206–241 (2004)
7. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) Advances in Cryptology - ASIACRYPT 2000, Proceedings. Lecture Notes in Computer Science, vol. 1976, pp. 531–545. Springer (2000)
8. Bertoni, G., Daemen, J., Hoffert, S., Peeters, M., Van Assche, G., Van Keer, R.: Farfalle: parallel permutation-based cryptography. IACR Trans. Symmetric Cryptol. 2017(4), 1–38 (2017)
9. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Duplexing the sponge: Single-pass authenticated encryption and other applications. In: Miri, A., Vaudenay, S. (eds.) Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7118, pp. 320–337. Springer (2011)
10. Boldyreva, A., Taesombut, N.: Online encryption schemes: New security notions and constructions. In: Okamoto, T. (ed.) Topics in Cryptology - CT-RSA 2004, The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23-27, 2004, Proceedings. Lecture Notes in Computer Science, vol. 2964, pp. 1–14. Springer (2004)
11. Bcuiei, N., Daemen, J., Hoffert, S., Van Assche, G., Van Keer, R.: Jammin' on the deck. IACR Cryptol. ePrint Arch. p. 531 (2022)
12. Chen, S., Steinberger, J.P.: Tight security bounds for key-alternating ciphers. In: Nguyen, P.Q., Oswald, E. (eds.) Advances in Cryptology - EUROCRYPT 2014, Proceedings. Lecture Notes in Computer Science, vol. 8441, pp. 327–350. Springer (2014)
13. Daemen, J., Hoffert, S., Peeters, M., Van Assche, G., Van Keer, R.: All on Deck! Real World Crypto 2020, New York, USA, January 8-10, 2020, https://rwc.iacr.org/2020/slides/Assche.pdf, https://www.youtube.com/watch?v=CQDsLhf-d-A (2020)
14. Daemen, J., Hoffert, S., Peeters, M., Van Assche, G., Van Keer, R.: Xoodyak, a lightweight cryptographic scheme. IACR Trans. Symmetric Cryptol. 2020(S1), 60–87 (2020)
15. Daemen, J., Hoffert, S., Van Assche, G., Van Keer, R.: The design of Xoodoo and Xoofff. IACR Trans. Symmetric Cryptol. 2018(4), 1–38 (2018)
16. Daemen, J., Massolino, P.M.C., Mehrdad, A., Rotella, Y.: The subterranean 2.0 cipher suite. IACR Trans. Symmetric Cryptol. 2020(S1), 262–294 (2020)
17. Daemen, J., Mennink, B., Van Assche, G.: Full-state keyed duplex with built-in multi-user support. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology - ASIACRYPT 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10625, pp. 606–637. Springer (2017)
18. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer (2002)
19. Daemen, J., Rijmen, V.: The pelican MAC function. IACR Cryptol. ePrint Arch. 2005, 88 (2005)
20. Duong, T., Rizzo, J.: Here come the XOR ninjas. Manuscript (2011)

21. Fouque, P., Joux, A., Martinet, G., Valette, F.: Authenticated on-line encryption. In: Matsui, M., Zuccherato, R.J. (eds.) Selected Areas in Cryptography, 10th Annual International Workshop, SAC 2003, Ottawa, Canada, August 14-15, 2003, Revised Papers. Lecture Notes in Computer Science, vol. 3006, pp. 145–159. Springer (2003)
22. Gunsing, A., Daemen, J., Mennink, B.: Deck-based wide block cipher modes and an exposition of the blinded keyed hashing model. IACR Trans. Symmetric Cryptol. 2019(4), 1–22 (2019)
23. Hamburg, M.: The STROBE protocol framework. In: Real World Crypto (2017)
24. Hoang, V.T., Krovetz, T., Rogaway, P.: Robust authenticated-encryption AEZ and the problem that it solves. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 15–44. Springer (2015)
25. Hoang, V.T., Reyhanitabar, R., Rogaway, P., Vizár, D.: Online authenticated-encryption and its nonce-reuse misuse-resistance. In: Gennaro, R., Robshaw, M. (eds.) Advances in Cryptology - CRYPTO, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9215, pp. 493–517. Springer (2015)
26. Hoang, V.T., Shen, Y.: Security of streaming encryption in google's tink library. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020. pp. 243–262. ACM (2020)
27. Iwata, T., Minematsu, K., Peyrin, T., Seurin, Y.: ZMAC: A fast tweakable block cipher mode for highly secure message authentication. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017, Proceedings, Part III. Lecture Notes in Computer Science, vol. 10403, pp. 34–65. Springer (2017)
28. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. Chapman and Hall/CRC Press (2007)
29. Liskov, M.D., Rivest, R.L., Wagner, D.A.: Tweakable block ciphers. In: Yung, M. (ed.) Advances in Cryptology - CRYPTO 2002, Proceedings. Lecture Notes in Computer Science, vol. 2442, pp. 31–46. Springer (2002)
30. Luby, M., Rackoff, C.: How to construct pseudorandom permutations from pseudorandom functions. SIAM J. Comput. 17(2), 373–386 (1988)
31. Lucks, S.: Faster Luby-Rackoff ciphers. In: Gollmann, D. (ed.) Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings. Lecture Notes in Computer Science, vol. 1039, pp. 189–203. Springer (1996)
32. Maurer, U.M., Renner, R., Holenstein, C.: Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In: Naor, M. (ed.) Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings. Lecture Notes in Computer Science, vol. 2951, pp. 21–39. Springer (2004)
33. Mennink, B., Neves, S.: Optimal PRFs from blockcipher designs. IACR Trans. Symmetric Cryptol. 2017(3), 228–252 (2017)
34. NIST: NIST special publication 800-38b, recommendation for block cipher modes of operation: the cmac mode for authentication (June 2016)
35. Patarin, J.: The "coefficients H" technique. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5381, pp. 328–345. Springer (2008)

36. Rogaway, P.: Authenticated-encryption with associated-data. In: Atluri, V. (ed.) Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002. pp. 98–107. ACM (2002)
37. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: Lee, P.J. (ed.) Advances in Cryptology - ASIACRYPT 2004, Proceedings. Lecture Notes in Computer Science, vol. 3329, pp. 16–31. Springer (2004)
38. Rogaway, P.: Nonce-based symmetric encryption. In: Roy, B.K., Meier, W. (eds.) Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers. Lecture Notes in Computer Science, vol. 3017, pp. 348–359. Springer (2004)
39. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: a block-cipher mode of operation for efficient authenticated encryption. In: Reiter, M.K., Samarati, P. (eds.) CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001. pp. 196–205. ACM (2001)
40. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) Advances in Cryptology - EUROCRYPT 2006, Proceedings. Lecture Notes in Computer Science, vol. 4004, pp. 373–390. Springer (2006)
41. Rogaway, P., Zhang, Y.: Simplifying game-based definitions - indistinguishability up to correctness and its application to stateful AE. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 3–32. Springer (2018)
42. Tsang, P.P., Solomakhin, R., Smith, S.W.: Authenticated streamwise on-line encryption. Dartmouth Computer Science Report TR2009-640 (2009)