Jammin' on the deck

Norica Băcuieți[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 modes with varying levels of robustness. Our modes have built-in support for sessions, but are also efficienty without them. As a by-product, we define a new ideal model for authenticated encryption 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.

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 block-cipher based 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 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 [22], and this is called PRF security.

Block cipher modes deal with dividing variable-length inputs into fixed-length blocks. This often comes with considerable complexity, such as dealing with complete last blocks, and this tends to propagate 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. [7] under the name of *deck function*. That paper presents a construction for building deck functions called *farfalle* and showcases an instance based on Keccak-f called Kravatte with excellent performance. Later the same authors presented a second farfalle instance called Xoofff improving on all aspects over Kravatte in [7]. 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!

Next to the simplicity of modes, performance is a clear and natural motivation for exploring authenticated encryption using deck functions. For instance, Kravatte and Xoofff have excellent reported performance figures and outperform modes using the AES block cipher, sometimes even when the platform has hardware AES support [5]. 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 [23] with underly-

[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 or the output is not the ciphertext.

ing block cipher AES-128 [9] and Pelican-MAC [10]: 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 a promising alternative to block-ciphers that is worth exploring.

Another motivation is that the incrementality feature of deck functions comes in handy, not only for the simplicity of authenticated encryption mode definitions, but also for the specific case of *session-supporting* authenticated encryption. 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. In this context, a *session* deals with the authentication of sequences of messages, preventing an attacker from reshuffling messages. Furthermore, ensuring that a message is authenticated in the context of previously sent messages comes essentially for free thanks to the incrementality properties of deck functions. Another interesting use case is the transmission of long messages to low-end devices, where intermediate tags can authenticate the message in an incremental way. In our exploration, we therefore consider authenticated encryption modes on top of deck functions, with or without support of sessions.

1.1 Our contributions

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

The ultimate ideal-world authenticated encryption scheme An ideal authenticated encryption scheme has at least one of the two following features:

1. *operational*: it can serve as an ideal SAE scheme to be used in higher-level protocols. It achieves the highest security thinkable (i.e., the cryptograms are as random as injectivity allows) while behaving deterministically (equal inputs give equal outputs under the same state).
2. *referential*: it can serve as an ideal-world model in distinguishability settings for SAE modes or schemes, to prove, or claim, a distinguishing bound.

We define in Section 2 the *jammin cipher* that combines both. This allows building a higher-level protocol that makes use of jammin cipher, prove it secure, and subsequently instantiate it with a concrete AE scheme. The security of the resulting protocol can then be quantified using the triangle inequality. Note that the jammin cipher naturally also covers non-session authenticated encryption.

For models that include the support of sessions, there do exist security definitions that we could use, namely, the Online Authenticated Encryption

(OAE2) security definitions [15]. Specifically, OAE2 covers streaming applications, where plaintexts and ciphertexts can be processed on the fly. However, they are unsuitable because 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. 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. This separation between the operational versions and the referential version leaves a security gap [15, Proposition 2] *larger than the one between the modes we define in this paper and our ideal scheme*, the jammin cipher.

Besides combining operational and referential roles in a single scheme, the jammin cipher has several features that make it superior 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 [15].
– 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. This is necessary when dealing with schemes that have variable ciphertext expansion due to the use of block encryption. Instead, OAE2 only supports ciphertext expansion by a fixed length.
– It addresses *multi-key security*.
– It supports unwrap and wrap calls in any order, including *bi-directional communication*. While the Ideal2B scheme is operational and supports authenticating multiple messages, an instance can only encipher messages or decipher cryptograms but not both.

The jammin cipher results from taking another look at OAE2 security, improving it and simplifying it.

Deck function-based session-supporting authenticated encryption 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 SAE 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. We summarize these modes in Table 1.

1.2 Notation

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

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 SAE modes.

is denoted as $X||Y$ and their bitwise addition as $X + Y$. Bit string values are noted with a typewriter font, such as `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.

1.3 Security setup

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.

2 The jammin cipher, an ideal-world SAE scheme

We define the *jammin cipher* in Algorithm 1.

2.1 Interface

We describe the jammin cipher in an object-oriented way, with *object instances* (or *instances* for short) held by the communicating parties. An instance belongs

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

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

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

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

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

15: Interface: inst.unwrap($A, C$) returns $P$ or $\perp$
16: context $\leftarrow$ inst.history; $A$
17: if $\exists! P$ : codebook(context; $P$) = $C$ then
18:     inst.history $\leftarrow$ inst.history; $A$; $P$
19:     return $P$
20: else
21:     taboo(context) $\leftarrow C$
22:     return $\perp$
```

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 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 $\text{WrapExpand}(p) = p + t$ with t some fixed length, e.g., 128 for stream encryption followed by a 128-bit tag. For OCB [27], we have $\text{WrapExpand}(p) = t \left\lceil \frac{p}{t} + 1 \right\rceil$ 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 $f\colon \mathbb{Z}_{\geq 0} \rightarrow \mathbb{Z}_{\geq 0}$ 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 $\mathsf{history}$. Initialization sets this attribute to the object identifier and each wrap and (successful) unwrap appends a message (A, P). So $\mathsf{history}$ 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 sequence of 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.

2.2 Inner workings

The jammin cipher keeps track of all wrap queries in a global archive called $\mathsf{codebook}$. This is a mapping from tuples $(\mathsf{history}; A; P)$ to a cryptogram or an error code. The data elements $\mathsf{history}$ 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 $\mathsf{codebook}$ 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 $\mathsf{codebook}$ are never overwritten, as the only place where a cryptogram value is assigned to $\mathsf{codebook}$ is on line 12, under the condition that $\mathsf{codebook}$ 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* $\mathsf{codebook}$ *one always recovers at most one plaintext value, i.e.,*

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

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 different 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 $\mathrm{WrapExpand}(|P|) = n$,
- 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 $n = \mathrm{WrapExpand}(0) \geq t$ 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 $n > \mathrm{WrapExpand}(0)$. 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 $\mathrm{WrapExpand}(p) = n$ 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. □

2.3 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.

2.4 Discussion

Deterministic wrapping has the limitation that it allows an adversary to tell identical plaintexts from identical cryptograms, and this can leak information. In particular, if the plaintext in the message comes from a set of small cardinality, an adversary can recover it from the cryptogram by wrapping the possible plaintexts. This opens to a family of attacks such as the chosen-prefix secret-suffix (CPSS) attack [11,15].

The countermeasure against these attacks is to make encryption *context-dependent*. If the user can ensure that the encryption context is different when identical plaintexts are encrypted, equal plaintexts will give different cryptograms and there is no leakage. The context for encryption is usually a message counter (e.g., in counter mode) or a (random) initial value (e.g., in CBC encryption) and, if it is unique for each encryption, we say it is a *nonce*. A data element that is unique per encryption is sometimes simply called a nonce, but this may turn out to be 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. Ultimately, whether the encryption context is a nonce depends on 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 proof of an upper bound on the distinguishing advantage between such a mode and the jammin cipher, proves that the leakage of the mode is limited to equality of plaintexts given equal cryptograms obtained with equal 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.

2.5 Bi-directional communication

The jammin cipher supports bi-directional communication. Alice and Bob, who wish to communicate securely, each create an instance of the jammin cipher with the same ID. They can subsequently encrypt their messages with wrap and decrypt/check them with unwrap, even sharing one session, i.e., having synchronized histories.

Figure 1 shows an example of a protocol between a client, Alice, and a server, Bob, who share one session. Alice and Bob each maintain a session counter. The session starts with Alice who wraps N_{Alice} as the associated data and no plaintext. Alice sends the cryptogram (or tag) T_0 to Bob, who checks it. In the rest of the protocol, Alice and Bob take turns at wrapping and unwrapping. At each step, $A.\mathsf{history}$ and $B.\mathsf{history}$ are equal.

As this protocol is run, the encryption context grows with all the previous associated data and plaintexts and is at all times a nonce. On Alice's side, she increments N_{Alice} after issuing the first wrap of the session; hence she will not start a new session with the same session counter. Bob stores in N_{Bob} the previous session counter, and assuming that the valid values of N_{Alice} are increasing, he accepts only $N_{\text{Alice}} > N_{\text{Bob}}$. When he calls wrap, he is therefore sure of using a fresh session counter. Note that Bob modifies N_{Bob} and synchronizes it with N_{Alice} only after checking T_0, therefore avoiding denial-of-service attacks where an adversary would control N_{Bob}.

This example highlights the power of the adversarial model that the jammin cipher represents. If such bi-directional communication protocol is secure in our model, then it is also secure with a concrete scheme that is infeasible to distinguish from the jammin cipher.

Fig. 1: Example of protocol. As a convention, every time there is a check, if it fails, we assume that the protocol ignores the messages or stops.

Alice		Bob
$\mathrm{ID}_{\mathrm{Alice\&Bob}}, N_{\mathrm{Alice}}$		$\mathrm{ID}_{\mathrm{Alice\&Bob}}, N_{\mathrm{Bob}}$
$N_{\mathrm{Alice}} \leftarrow N_{\mathrm{Alice}} + 1$		
$A \leftarrow \mathsf{init}(\mathrm{ID}_{\mathrm{Alice\&Bob}})$		
$T_0 \leftarrow A.\mathsf{wrap}(N_{\mathrm{Alice}}; \epsilon)$	$\xrightarrow{\mathrm{Alice}, N_{\mathrm{Alice}}, T_0}$	$N_{\mathrm{Alice}} \overset{?}{>} N_{\mathrm{Bob}}$
		$B \leftarrow \mathsf{init}(\mathrm{ID}_{\mathrm{Alice\&Bob}})$
		$B.\mathsf{unwrap}(N_{\mathrm{Alice}}, T_0) \overset{?}{\neq} \perp$
		$N_{\mathrm{Bob}} \leftarrow N_{\mathrm{Alice}}$
$P_1 \leftarrow A.\mathsf{unwrap}(\epsilon, C_1) \overset{?}{\neq} \perp$	$\xleftarrow{C_1}$	$C_1 \leftarrow B.\mathsf{wrap}(\epsilon, \text{“Today's menu”})$
$C_2 \leftarrow A.\mathsf{wrap}(\epsilon, \text{“a command”})$	$\xrightarrow{C_2}$	$P_2 \leftarrow B.\mathsf{unwrap}(\epsilon, C_2) \overset{?}{\neq} \perp$
$P_3 \leftarrow A.\mathsf{unwrap}(\epsilon, C_3) \overset{?}{\neq} \perp$	$\xleftarrow{C_3}$	$C_3 \leftarrow B.\mathsf{wrap}(\epsilon, \text{“a response”})$
	etc.	

With codebook and taboo being global variables, the ideal model is omniscient. Even if Alice and Bob are distant, the responses of their calls to wrap and unwrap remain deterministic and consistent. In a real-world scheme, it simply means that Alice and Bob share the same secret key and therefore can compute exactly the same things. Also, if Alice or Bob tries to unwrap a message that none of them wrapped, an error is inevitably returned. In a real-world scheme, checking the tag achieves the same goal except for the probability $p \leq 2^{-t}$ that the adversary correctly guessed it.

For simplicity, we assume that Alice, as the client, always initiates the communication. This avoids that Alice and Bob initiate the communication with the same value N. Alternatively, Alice and Bob could maintain a pair of session counters N_{Alice} and N_{Bob} in separate domains (e.g., N_{Alice} is always even and N_{Bob} is always odd) so as to authenticate the initiator of the communication.

2.6 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 [15, Fig. 6] (see also Appendix C), 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 Appendix B.1.

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 [15, 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 as input 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 ([7]). *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 detailed in Section 3.1, and typically comes with a pseudorandomness security claim, see Section 3.2.

Regarding the notation, we assume that the number of bits that the deck function outputs is determined by the context, and in particular makes the length of two strings involved in a bitwise addition equal. 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 Incrementality

A deck function should allow efficient incremental computing. In particular, by keeping state after computing an output for input sequence $X = X^{(0)}; \ldots; X^{(m-1)}$, computing an output for $X; Y^{(0)}; \ldots; Y^{(n-1)}$ should have a cost independent of X. In addition, by keeping state after computing $\mathsf{0}^n + F_K\left(X^{(0)}; \ldots; X^{(m-1)}\right) \ll q$, computing $\mathsf{0}^m + F_K\left(X^{(0)}; \ldots; X^{(m-1)}\right) \ll (q+n)$ should have a cost independent of n or q.

More formally, we assume that a deck function F can be implemented by defining a finite state set $\mathcal{S}$ and three auxiliary functions:

- $\mathsf{Init}_F : \mathcal{K}_F \to \mathcal{S}$. Calling $s \leftarrow \mathsf{Init}_F(K)$ processes the key $K \in \mathcal{K}_F$ and returns the initial state s.
- $\mathsf{Input}_F : \mathcal{S} \times \mathbb{Z}_2^* \to \mathcal{S}$. Calling $s \leftarrow \mathsf{Input}_F(s, X)$ processes the string X and updates the state s.
- $\mathsf{Output}_F : \mathcal{S} \times \mathbb{N} \to \mathbb{Z}_2^* \times \mathcal{S}$. Calling $(Z, s) \leftarrow \mathsf{Output}_F(s, n)$ returns the output string $Z \in \mathbb{Z}_2^n$ and updates the state s.

The output string returned by $\mathsf{Output}_F(s, n)$ is the output of the deck function $F_K(\ldots)$ with K processed by the last call to Init_F and the sequence of strings processed by the calls to Input_F. Consecutive calls to Output_F extend the outputs produced since the last call to Input_F. As an example, consider the following sequence of calls.

$$\begin{aligned}
s &\leftarrow \mathsf{Init}_F(K) & & \\
s &\leftarrow \mathsf{Input}_F(s; X^{(0)}) & & \\
(Z_1, s) &\leftarrow \mathsf{Output}_F(s, n_1) & Z_1 &= \mathsf{0}^{n_1} + F_K(X^{(0)}) \\
(Z_2, s) &\leftarrow \mathsf{Output}_F(s, n_2) & Z_2 &= \mathsf{0}^{n_2} + F_K(X^{(0)}) \ll n_1 \\
s &\leftarrow \mathsf{Input}_F(s; X^{(1)}) & & \\
(Z_3, s) &\leftarrow \mathsf{Output}_F(s, n_3) & Z_3 &= \mathsf{0}^{n_3} + F_K(X^{(0)}; X^{(1)})
\end{aligned}$$

The incrementality of the deck function requires that $\mathsf{Input}_F(s, X)$ and $\mathsf{Output}_F(s, n)$ take a time that depends only on $|X|$ and n, respectively, and that is independent of s, although block effects are still possible.

3.2 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.

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

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

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.

3.3 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 [8] and farfalle [3]. For the former, we can mention Strobe [13] and Xoodyak [6] as concrete instantiations. For the latter, Kravatte [3] and Xoofff [7] are two farfalle instantiations making use of the Keccak-f and Xoodoo permutations respectively.

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 [18]. First, we compress the input through a secure MAC construction such as PMAC1 [25] or ZMAC [16], 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) authenticated encryption 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\$ [26]. This shows the following proposition:

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

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

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 $T \leftarrow 0^t + F_K(P)$. One can verify a tag by taking as input the message P and its tag T and check whether $T + F_K(P)$ equals 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 [17, 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*

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

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, authenticated encryption 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 $T \leftarrow 0^t + F_K(A; Z)$. 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 SAE 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 and an alignment unit length ℓ. The former determines the security level, while the latter 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 of strings. Every call of the deck function absorbs the full history but as it is efficiently incremental, only the strings that were appended since previous deck function call need to be processed.

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.

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 $\ell$ not smaller than $t$

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

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

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

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

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). Hence, the individual calls to (un)wrap can be identified in the history without ambiguity.

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 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.

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*

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

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

We start by introducing the H-coefficient technique and defer the proof to Section 4.4.

4.3 The H-coefficient technique

Our proofs use the H-coefficient technique from Patarin [24]. We will follow the adaptation of Chen and Steinberger [4]. 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 [4].

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 $\tau \in \mathcal{T}_{\text{good}}$:* $\Pr(D_{\mathcal{O}} = \tau) / \Pr(D_{\mathcal{P}} = \tau) \geq 1 - \varepsilon$ *. Then,* $\Delta_{\mathcal{A}}(\mathcal{O}\ ;\ \mathcal{P}) \leq \varepsilon + \Pr(D_{\mathcal{P}} \in \mathcal{T}_{\text{bad}})$*.*

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 $\tau \in \mathcal{T}_{\text{good}}$, so that $\Delta_{\mathcal{A}}(\mathcal{O}\ ;\ \mathcal{P}) \leq \Pr(D_{\mathcal{P}} \in \mathcal{T}_{\text{bad}})$, 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.,

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

where $\mathcal{D}''$ has the same resources as $\mathcal{D}$.

We then use Lemma 1 with $\mathcal{O} = \mathcal{J}^{+t} \triangleq \mathcal{J}$ and $\mathcal{P} = \text{Deck-PLAIN}(\mathcal{RO}, t, \ell)$. 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

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

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. We use this to have 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 $\mathsf{context}$ 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 $\mathsf{context}$ value.

As the $\mathsf{context}$ 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, $\mathcal{P} = \text{Deck-PLAIN}(\mathcal{RO}, t, \ell)$ 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 $Z = P + \mathcal{RO}(\mathsf{context})$, $\mathsf{context} = \text{history}; A$ (or $P + \mathcal{RO}(\text{history}) \ll \text{offset}$ 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 random oracle output stream, it contributes a factor exactly $2^{-(|P|+t)}$ to $\Pr(D_{\mathcal{P}} = \tau)$. A wrap record with $P = \epsilon$ contribute a factor exactly 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. $\square$

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 [2]).

After contemplating different modes like Synthetic Initial Value (SIV) [28], Robust IV (RIV) [1] and wide-block ciphers [19, 20], it occurs that they can all be expressed under the hood of a Feistel network. We here give an intuitive overview of these modes from this point of view, starting with the simplest case: SIV. Consider Figure 2 (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 decrypted ciphertexts 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

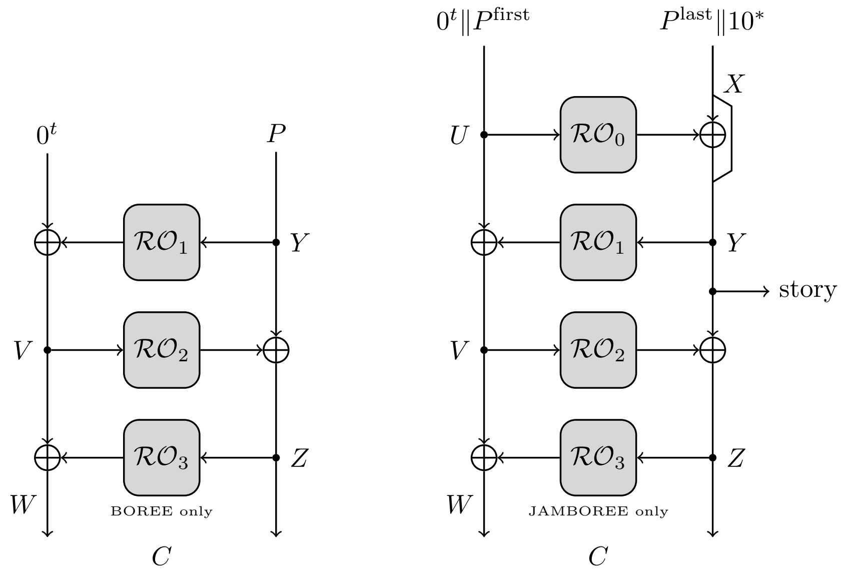

Fig. 2: Feistel network inside the different modes, Deck-BO(REE) on the left and Deck-JAMBO(REE) on the right.

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 2 (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 an improved SIV mode, with the 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.

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** $\mathsf{jam} \in$ flags **then**
    $R_0 \leftarrow R_0 + F_K\,(\mathsf{context}; L||\mathsf{001})$
$L \leftarrow L + F_K\,(\mathsf{context}; R||\mathsf{011})$
history $\leftarrow \mathsf{context}; R||\mathsf{011}$
$R \leftarrow R + F_K\,(\mathsf{context}; L||\mathsf{101})$
**if** $\mathsf{ree} \in$ flags **then**
    $L_0 \leftarrow L_0 + F_K\,(\mathsf{context}; R||\mathsf{111})$
**return** (history, $L||R$)

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

5.1 Deck-BO

Deck-BO, defined in Algorithm 4, combines the SIV approach [28] 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.

In contrast to Deck-PLAIN, the leakage of Deck-BO is limited to revealing equality of plaintexts given equal encryption contexts. To achieve that, Deck-BO first 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.

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

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

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*

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

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 $\mathsf{context}$ *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 $\frac{q_{\mathsf{wrap}}^2}{2^{t+1}}$ and we have to 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 is similar to that of Theorem 6, except that (i) V, Y and $U||X$ are not added to the records, (ii) the bad events are only the successful forgery and the tag collision. The complete proof can be found in Appendix B.2.

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 [21]. 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*

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

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 contribute to $\sigma'(\mathsf{context})$ and we get the same bound as in Theorem 3 for Deck-BO.

The proof is similar to that of Theorem 6, except that (i) $U||X$ does not need to be added to the records and Y is added only to unwrap records, (ii) the bad events are only the successful forgery and the hidden tag (or V) collision. The complete proof can be found in Appendix B.3.

5.3 Deck-JAMBO and optimal redundancy

Deck-JAMBO can be seen as an enhancement of Deck-BO to allow mixing plaintext with redundancy, resulting in less required expansion. This is accomplished by performing a 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.

For security, we require the plaintext expansion and the splitting of the input to satisfy some properties. The split cuts the expanded plaintext or cryptogram into four parts, as the left and right parts are further split for the optional first and last rounds. We formalize this with three functions, 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 $\mathtt{0}^t||P||\mathtt{10}^*$. 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 WrapExpand$(0) = t$ and therefore the implicitly defined WrapExpand function is t-expanding by construction.

Second, we define a plaintext extraction function called extract(P', t) that returns $\perp$ if P' does not start with $\mathtt{0}^t$ 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.

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

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

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* $\text{Deck-JAMBO}(F, t, \text{expand}, \text{split})$ *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 exists 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 $\mathsf{context}$ *value.*

The proof is similar to that of Theorem 6, except that (i) V and $U||X$ are not added to the records and Y only to wrap records, (ii) the bad events are only the successful forgery, and the left branch (or V) collision and the plaintext representative (or Y) collision. The complete proof can be found in Appendix B.4.

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 [14] but rather specified in terms of a deck function, like Doubledecker [12]. 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* $\text{Deck-JAMBOREE}(F, t, \text{expand}, \text{split})$ *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*

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

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

Proof. As done in the proof of Theorem 2, we use a hybrid argument and replace the deck function with a random oracle before comparing Deck-JAMBOREE with the jammin cipher. We then use Lemma 1 with $\mathcal{O} = \mathcal{J}$ and $\mathcal{P} =$ Deck-JAMBOREE$(\mathcal{RO}, \dots)$.

In this proof, we use the session syntax of the jammin cipher. The context is $\mathsf{history}; A$ in the jammin cipher, or story$; A||\mathsf{10}$ (or just story if $A = \epsilon$) in Deck-JAMBOREE. However, in Deck-JAMBOREE, what is kept in story is not the plaintext but the *plaintext representative* Y. This mapping is not injective but collisions are rare and we treat them as bad events. If no bad events happen, the story is an injective function of $\mathsf{history}$. For additional uniformity in the notation, $\mathcal{RO}_0(\mathsf{context}; X)$ (resp, $\mathcal{RO}_1$, etc.) is a call to $\mathcal{RO}(\mathsf{context}; X||\mathsf{001})$ (resp. $\mathcal{RO}(\mathsf{context}; X||\mathsf{011})$, etc.) in Deck-JAMBOREE.

The plaintext representative Y is computed as follows. Let $(L_0, L_+, R_0, R_+) = \text{split}(\text{expand}(P, t), t)$ and denote the lengths of the components as $l = |L_0| + |L_+|$, $r_0 = |R_0|$ and $r_+ = |R_+|$. Because of expand(P, t), we have that $L_0||L_+ = \mathsf{0}^t||P^{\text{first}}$ and $R_0||R_+ = P^{\text{last}}||\mathsf{10}^{\text{pad}}$, with P split as $P = P^{\text{first}}||P^{\text{last}}$ accordingly. Then,

$$\begin{aligned} Y &= (R_0||R_+) + (\mathcal{RO}_0(\mathsf{context}; L_0||L_+)||\mathsf{0}^{r_+}) \\ &= (P^{\text{last}}||\mathsf{10}^{\text{pad}}) + (\mathcal{RO}_0(\mathsf{context}; \mathsf{0}^t||P^{\text{first}})||\mathsf{0}^{r_+})\,. \end{aligned} \tag{1}$$

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

$$(\mathsf{wrap}, \mathsf{context}, P, C, V, Y),$$
$$(\mathsf{unwrap}, \mathsf{context}, P, C, V, Y, U||X),$$

where the first component indicates the type of call made and the second contains the context in an agnostic representation. Also, $\mathsf{context} = \text{history}; A$ when $P = \epsilon$. For a call to wrap, P is its parameter and C is the returned value, with $C \neq \perp$. For a call to unwrap, C is its parameter and P, the plaintext or an error code, is the returned value. 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. We use this to have a simple definition of forgery, namely as a successful unwrap record.

There are additional components in the records.

1. Y, the plaintext representative in wrap records. Its presence is required to define bad events, and it is disclosed only at the end of the game. When $P = \epsilon$, we have $Y = \epsilon$.
   – In Deck-JAMBOREE, Y is the right branch of B after the first round, as in Equation (1) (see also Figure 2).

- In the jammin cipher, $Y = P^{\text{last}} || 10^{\text{pad}} + \mathcal{RO}_0(\mathsf{context}; P^{\text{first}}) || 0^{r+}$, with $\mathcal{RO}_0$ being statistically independent of the coins used in the jammin cipher.

2. V, the hidden diversifier V in wrap and unwrap records. When $P = \epsilon$ or $|C| = t$, $V = \epsilon$. Its presence is required to define bad events, and it is disclosed only at the end of the game. In both cases it is computed from parts of the cryptogram C. Let $W||Z = C$ with $|W| = l$, see also Figure 2.
   - In Deck-JAMBOREE, $V = W + \mathcal{RO}_3(\mathsf{context}; Z)$.
   - In the jammin cipher, $V = W + \mathcal{RO}_3(\mathsf{context}; Z)$, with $\mathcal{RO}_3$ statistically independent of the coins used in the jammin cipher.
3. Y, the unverified plaintext representative in unwrap records. Y is used to define bad events, and it is disclosed only at the end of the game. $Y = \perp$ when the adversary does not access the unverified decrypted cryptogram.
   - In Deck-JAMBOREE, $Y = Z + \mathcal{RO}_2(\mathsf{context}; V)$.
   - In the jammin cipher, $Y = X + \mathcal{RO}_0(\mathsf{context}; U) || 0^{r+}$ for consistency. $\mathcal{RO}_0$ is independent of the coins used in the jammin cipher and $U||X$ is determined as detailed below.
4. $U||X$, the unverified decrypted cryptogram in unwrap records. This data element may be disclosed upon each unwrap query so that the adversary can react adaptively. In all cases, $U||X = Y = \perp$ when the adversary does not access the unverified decrypted cryptogram.
   - In Deck-JAMBOREE, $U = V + \mathcal{RO}_1(\mathsf{context}; Y)$ and $X = Y + \mathcal{RO}_0(\mathsf{context}; U) || 0^{r+}$. In the case of a valid unwrap query, $U||X$ would contain the actual plaintext.
   - In the jammin cipher, $U||X = 0^{|C|} + \mathcal{RO}_2(\mathsf{context}; C)$ and $\mathcal{RO}_2$ is statistically independent from $\mathcal{RO}_0$ and independent of the coins used in the jammin cipher, so Y bears no relation with any plaintext.

We have 3 types of bad event. If a transcript has a bad event, it is in $\mathcal{T}_{\text{bad}}$.

Collision in plaintext representative Y : the transcript has two distinct records of the form $(\mathsf{wrap}, \mathsf{context}, P, C, V, Y)$ with $P \neq \epsilon$ or $(\mathsf{unwrap}, \mathsf{context}, P, C, V, Y \neq \perp, U||X)$ with equal context and equal value Y. For the wrap queries to Deck-JAMBOREE, the first r_0 ($\geq 2t$) bits of Y are randomly generated with $\mathcal{RO}$. As Y is disclosed only at the end of the game, the adversary cannot use P^{last} adaptively to compensate for changes in $\mathcal{RO}(\ldots; P^{\text{first}}|| \ldots)$ and obtain a collision. Also, when the adversary varies P^{last} but not P^{first} the resulting Y cannot collide. For the unwrap queries to Deck-JAMBOREE, $Y = Z + \mathcal{RO}_2(\mathsf{context}; V)$ and distinct queries with colliding V's cannot lead to collisions in Y. So in the end, we look at the probability of collisions on strings of at least $2t$ random bits among at most $\sigma'(\mathsf{context})$ records at once.

Collision in hidden left branch V : the transcript has two distinct records of the form $(\mathsf{wrap}, \mathsf{context}, P, C, V, Y)$ or $(\mathsf{unwrap}, \mathsf{context}, P, C, V, Y, U||X \neq \perp)$ with equal context, $|C| > t$ and equal values V. For wrap queries to Deck-JAMBOREE, $V = 0^t || P^{\text{first}} + \mathcal{RO}_1(\mathsf{context}; Y)$ and Y can be assumed to be unique per query. For unwrap queries to Deck-JAMBOREE, V is computed from $C = W||Z$, noting that 1) the values of V cannot be used adaptively to adjust W and obtain a collision and 2) when the adversary varies W but not Z the resulting V cannot collide. In addition, collisions between wrap and unwrap queries must also be taken into account, and again we look at the probability of collisions on strings of at least $2t$ random bits among at most $\sigma'(\mathsf{context})$ records at once.

Successful forgery : the transcript has a record of the form $(\mathsf{unwrap}, \mathsf{context}, P, C, V, U||X)$ with $P \neq \perp$. After q_{unwrap} queries, the probability that the left branch starts with t zeroes for one of the queries is upper bounded by $q_{\mathsf{unwrap}} 2^{-t}$. This is true even if different queries lead to identical values at the input of $\mathcal{RO}_1$.

This yields $\Pr(D_{\mathcal{P}} \in \mathcal{T}_{\text{bad}}) \leq \frac{q_{\mathsf{unwrap}}}{2^t} + \sum_{\mathsf{context}} \frac{\binom{\sigma'(\mathsf{context})}{2}}{2^{2t-1}}$.

We now argue 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, cryptogram and additional component bits are generated randomly and independently for different contexts, so we can partition the transcript records per $\mathsf{context}$ 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 $\mathsf{context}$ value. In such records, there can be several records of the forms $(\mathsf{wrap}, \mathsf{context}, P, C, V, Y)$ and $(\mathsf{unwrap}, \mathsf{context}, \perp, C, V, Y, U||X)$.

In the jammin cipher, each wrap record with $P \neq \epsilon$ and a new value of P^{first} contributes a factor 2^{-r_0} due to the value of Y. The value of each C in the wrap queries is taken from a set of cardinality at most $2^{|C|}$. Hence, each such record contributes a factor at least $2^{-|C|}$ to $\Pr(D_{\mathcal{J}} = \tau)$, conditioned on the previously considered records. Conditioned on the wrap records, the unwrap records contribute only through their values V, Y and $U||X$ (when $U||X \neq \perp$) because the jammin cipher avoids all forgeries. As $U||X = 0^{|C|} + \mathcal{RO}_2(\mathsf{context}; C)$, this contributes a factor $2^{-|C|}$ exactly. Since $Y = X + \mathcal{RO}_0(\mathsf{context}; U)||0^{r_+}$, this contributes a factor 2^{-r_0} for a fresh value U and 1 for repeating values U conditioned on the first occurrence with U. Hence, V, Y and $U||X$ in unwrap queries contribute a factor $2^{-|C|-r_0}$ or $2^{-|C|}$ depending on the repetitions of U. As $V = W + \mathcal{RO}_3(\mathsf{context}; Z)$, wrap and unwrap queries with a fresh Z contribute 2^{-l}, while those that vary only on W contribute 1 conditioned on the first query with Z. The jammin cipher may return an error, but thanks to Proposition 2, this would require $q_{\mathsf{unwrap}} \geq 2^t$.

In $\mathcal{P}$ = Deck-JAMBOREE($\mathcal{RO}, \dots$), each wrap record with $P \neq \epsilon$ and a new value of P^{first} contributes a factor 2^{-r_0} due to the value of Y. The value of each $C = W||Z$ in the wrap queries is obtained as $Z = Y + \mathcal{RO}_2(\mathsf{context}; V)$ and $W = 0^t||P^{\text{first}} + \mathcal{RO}_1(\mathsf{context}; Y) + \mathcal{RO}_3(\mathsf{context}; Z)$. Given that all the V and Y values are distinct in $\mathcal{T}_{\text{good}}$, the contribution of C is a factor exactly $2^{-|C|}$ to $\Pr(D_{\mathcal{P}} = \tau)$. Conditioned on the wrap records, we look at the contributions of the values of V, Y and $U||X$ (when $U||X \neq \perp$) in the unwrap records. The contribution of $P = \perp$ amounts to at most 1 in $\Pr(D_{\mathcal{P}} = \tau)$ as forgeries may happen. As $Y = Z + \mathcal{RO}_2(\mathsf{context}; V)$, the value Y contributes a factor 2^{-r} since the V's are distinct. As $U = V + \mathcal{RO}_1(\mathsf{context}; Y)$, the value U contributes a factor 2^{-l} since the Y's are distinct. As $X = Y + \mathcal{RO}_0(\mathsf{context}; U)||0^{r+}$, the value X in queries with a fresh U contribute a factor 2^{-r_0}, while the others contribute 1 conditioned on the first query with U. Hence, V, Y and $U||X$ in unwrap queries contribute a factor $2^{-|C|-r_0}$ or $2^{-|C|}$ depending on the repetitions of U. For wrap queries, $V = 0^t + \mathcal{RO}_1(\mathsf{context}; Y)$, which contributes a factor exactly 2^{-l} since all the Y's are distinct. For unwrap queries, $V = W + \mathcal{RO}_3(\mathsf{context}; Z)$ and queries with a fresh Z contribute 2^{-l}, while those that vary only on W contribute 1 conditioned on the first query with Z.

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

6 Conclusions

In this paper, we proposed the jammin cipher as an ideal model for session-supporting authenticated encryption (SAE), with a number of advantages over OAE2. Also, we investigated the use of deck functions to build SAE schemes with various robustness properties, and we analyzed their security under the jammin cipher model.

Deck functions provide an efficient and simple way of building SAE schemes. 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.

References

1. 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), https://doi.org/10.1007/978-3-662-52993-5_2
2. 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 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I. Lecture Notes in Computer Science, vol. 8873, pp. 105–125. Springer (2014), https://doi.org/10.1007/978-3-662-45611-8_6
3. 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), https://tosc.iacr.org/index.php/ToSC/article/view/801
4. Chen, S., Steinberger, J.P.: Tight security bounds for key-alternating ciphers. In: Nguyen, P.Q., Oswald, E. (eds.) Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications

of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8441, pp. 327–350. Springer (2014), https://doi.org/10.1007/978-3-642-55220-5_19
5. 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)
6. 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), https://doi.org/10.13154/tosc.v2020.iS1.60-87
7. 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), https://doi.org/10.13154/tosc.v2018.i4.1-38
8. 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 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10625, pp. 606–637. Springer (2017), https://doi.org/10.1007/978-3-319-70697-9_21
9. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer (2002), https://doi.org/10.1007/978-3-662-04722-4
10. Daemen, J., Rijmen, V.: The pelican MAC function. IACR Cryptol. ePrint Arch. 2005, 88 (2005), http://eprint.iacr.org/2005/088
11. Duong, T., Rizzo, J.: Here come the XOR ninjas. Manuscript (2011)
12. 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), https://doi.org/10.13154/tosc.v2019.i4.1-22
13. Hamburg, M.: The STROBE protocol framework. In: Real World Crypto (2017)
14. 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 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 15–44. Springer (2015), https://doi.org/10.1007/978-3-662-46800-5_2
15. 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), https://doi.org/10.1007/978-3-662-47989-6_24
16. 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 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III. Lecture Notes in Computer Science, vol. 10403, pp. 34–65. Springer (2017), https://doi.org/10.1007/978-3-319-63697-9_2
17. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. Chapman and Hall/CRC Press (2007), http://www.cs.umd.edu/%7Ejkatz/imc.html
18. Liskov, M.D., Rivest, R.L., Wagner, D.A.: Tweakable block ciphers. In: Yung, M. (ed.) Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryp-

tology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2442, pp. 31–46. Springer (2002), https://doi.org/10.1007/3-540-45708-9_3

19. Luby, M., Rackoff, C.: How to construct pseudorandom permutations from pseudorandom functions. SIAM J. Comput. 17(2), 373–386 (1988), https://doi.org/10.1137/0217022
20. 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), https://doi.org/10.1007/3-540-60865-6_53
21. 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), https://doi.org/10.1007/978-3-540-24638-1_2
22. Mennink, B., Neves, S.: Optimal PRFs from blockcipher designs. IACR Trans. Symmetric Cryptol. 2017(3), 228–252 (2017), https://doi.org/10.13154/tosc.v2017.i3.228-252
23. NIST: NIST special publication 800-38b, recommendation for block cipher modes of operation: the cmac mode for authentication (June 2016)
24. 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), https://doi.org/10.1007/978-3-642-04159-4_21
25. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: Lee, P.J. (ed.) Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3329, pp. 16–31. Springer (2004), https://doi.org/10.1007/978-3-540-30539-2_2
26. 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), https://doi.org/10.1007/978-3-540-25937-4_22
27. 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), https://doi.org/10.1145/501983.502011
28. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4004, pp. 373–390. Springer (2006), https://doi.org/10.1007/11761679_23

A The jammin cipher step-by-step

In this section, we build the jammin cipher incrementally. We describe three intermediate ideal AE schemes, and each time correct a flaw or add a feature. The purpose of this section is to serve as a tutorial and to give a different perspective on the jammin cipher as specified in Algorithm 1.

A.1 Conventions

We describe the jammin cipher and the intermediate schemes 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 groups that make use of the AE scheme simultaneously. For example, Alice and Bob may secure their communication each using an instance 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 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.

A.2 Initial requirements

Our scheme shall satisfy the following requirements:

Deterministic wrapping: An object wraps equal messages (A, P) to equal cryptograms C.

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

Forgery-freeness: An object will only unwrap successfully cryptograms C resulting from prior wrap calls with the same A.

We give an encryption scheme that satisfies these three requirements in Algorithm 6. It satisfies determinism and fully prevents forgery by keeping track of all wrap queries in an archive called codebook. This is a mapping from tuples $(\mathrm{ID}; A; P)$ to a cryptogram or an error code.

Initially, all the entries of codebook return an error. The expression

$$\mathsf{codebook}(\mathrm{ID}; A; P) \xleftarrow{\$} \mathcal{S}$$

Algorithm 6 jammin cipher Mk I

```
Global variables: codebook initially set to $\perp$ for all

Instance constructor: init(ID)
return new instance inst with attribute inst.id = ID
(In the sequel, we use ID to mean the object identifier saved in inst.id.)

Interface: inst.wrap$(A, P)$ returns $C$
if codebook$(\mathrm{ID}; A; P) = \perp$ then codebook$(\mathrm{ID}; A; P) \xleftarrow{\$} \mathbb{Z}_2^{|P|}$
return codebook$(\mathrm{ID}; A; P)$

Interface: inst.unwrap$(A, C)$ returns $P$ or $\perp$
if $\exists ! P$ : codebook$(\mathrm{ID}; A; P) = C$ then return $P$
else return $\perp$
```

denotes the assignment of a random element chosen uniformly from $\mathcal{S}$ to the entry codebook$(\mathrm{ID}; A; P)$.

The jammin cipher Mk I satisfies the three requirements:

– Deterministic wrapping: when wrapping a plaintext P for equal ID and A that was wrapped before, the cipher just returns the stored codebook entry.
– Random cryptograms: wrapping a message of $|P|$ bits results in a ciphertext randomly chosen from the set of $|P|$-bit strings.
– Forgery-free: all valid cryptograms are in the codebook and unwrapping a cryptogram not in the codebook results in an error.

A.3 Adding injectivity and determinism

Our scheme in Algorithm 6 has two non-ideal properties:

1. Wrapping is not injective: An object may wrap messages with equal A but different plaintexts to equal cryptograms. If that happens, unwrapping of the cryptogram fails as there are multiple possible plaintexts.
2. Unwrapping is not deterministic: An invalid cryptogram may become valid later if the object that unsuccessfully tried to unwrap it for some A outputs it as a cryptogram to a wrap call with the same A.

Both problems can be solved by excluding certain strings when generating the cryptogram values. Injectivity is achieved if the object excludes cryptogram values that it returned in earlier wrap calls for the same A. Deterministic unwrapping is achieved if the object excludes cryptogram values that it returned an error to in earlier unwrap calls for the same A. We keep track of the latter in an archive called taboo, that is a mapping from tuples $(\mathrm{ID}; A)$ to sets of cryptograms. We implemented this in Algorithm 7. The expression codebook$(\mathrm{ID}; A; *)$ denotes the set of values codebook$(\mathrm{ID}; A; P)$ over all P.

Our ideal-world AE scheme now satisfies the two additional requirements:

Injective wrapping: An object wraps messages with equal A and different P to different cryptograms.

Deterministic unwrapping: An object unwraps equal cryptograms and A to equal responses.

Algorithm 7 jammin cipher Mk II

```
Global variables: codebook initially set to $\bot$ for all, taboo initially set to *empty*

Instance constructor: init(ID)
return new instance inst with attribute inst.id = ID
(In the sequel, we use ID to mean the object identifier saved in inst.id.)

Interface: inst.wrap($A, P$) returns $C$
if codebook(ID; $A$; $P$) = $\bot$ then
    $\mathcal{C} = \mathbb{Z}_2^{|P|} \setminus$ (codebook(ID; $A$; $*$) $\cup$ taboo(ID; $A$))
    if $\mathcal{C} = \varnothing$ then return $\bot$
    codebook(ID; $A$; $P$) $\xleftarrow{\$} \mathcal{C}$
return codebook(ID; $A$; $P$)

Interface: inst.unwrap(ID, $A, C$) returns $P$ or $\bot$
if $\exists! P$ : codebook(ID; $A$; $P$) = $C$ then return $P$
else taboo(ID; $A$) $\leftarrow C$ and return $\bot$
```

A.4 Expanding the ciphertexts

We see in Algorithm 7 that satisfying these two additional requirements results in a limitation: the space of possible cryptograms may become very small resulting in predictable cryptograms. It can even be exhausted, resulting in failure to wrap due to lack of *free cryptogram values*. For n-bit plaintexts, this would be the case after $2^n + 1$ calls to wrap and unwrap of an object with the same A. As there is no lower bound on plaintext length, for $n = 1$ this can be as little as 3.

The property that is at the root of this problem is that for any object and A, all cryptogram values up to some length can be made valid by just making wrap calls or invalid by making unwrap calls. In real-world authenticated encryption schemes this is not possible because valid cryptogram values are *rare*: For an object and given some A and cryptogram length, valid cryptograms form a negligible subset of the space of all possible cryptograms. Ensuring that for our ideal-world AE scheme valid cryptograms are rare, we have to adopt a solution where wrapping is length-increasing: C is longer than P.

As increasing the length can be done in multiple ways, we parameterize our scheme by a function that specifies the length of the cryptogram given the length of the plaintext: WrapExpand(p). 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, we have $\mathrm{WrapExpand}(p) = t\left\lceil \frac{p}{t} + 1 \right\rceil$ with t the block length of the cipher. Clearly, exhausting all cryptograms corresponding to plaintexts of length p for a given object and A now requires at least $2^{\mathrm{WrapExpand}(p)}$ calls to unwrap (or part of them to wrap).

We specify the resulting ideal-world AE scheme in Algorithm 8.

Algorithm 8 jammin cipher Mk III

```
Parameter: WrapExpand
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.id = ID
(In the sequel, we use ID to mean the object identifier saved in inst.id.)

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

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

A.5 Diversification

Algorithms 6-8 have the limitation that an object will wrap equal messages to equal cryptograms. Deterministic wrapping has the side-effect of allowing an adversary to tell identical plaintexts from identical ciphertexts, and this can leak information. In particular, if the plaintext comes from a set of small cardinality, an adversary can recover it from the ciphertext by wrapping the possible plaintexts. This opens to a family of attacks such as the chosen-prefix secret-suffix (CPSS) attack [11, 15].

To overcome this, it is up to the user to ensure that the associated data A is always different. A data element that is guaranteed to be unique per call to wrap is called a *nonce*, so the associated data A shall be a nonce.

In addition to avoid the pitfalls of deterministic encryption, it turns out that one can build AE modes requiring the associated data A to be a nonce, e.g., by including a message counter or some other unique data elements, that are more efficient than those not requiring it.

Algorithm 8 can serve as a security reference for modes in use cases where A is not a nonce. Typically, a mode that requires A to be a nonce will be proven indistinguishable from the ideal-world model under the condition that A is a nonce.

Ultimately, it is up to the caller to ensure A uniqueness. Note that diversifiers do not have to be unique for wrap calls with empty plaintexts, that is, purely authentication requests. This follows from the idea that there cannot be any leakage from such calls since they do not encrypt anything. For instance, in a stream encryption mode, this would not reveal the keystream.

A.6 Supporting sessions

Finally, we arrive at the jammin cipher. So far, the *context* in which encryption was taking place was the ID; A pair. In this fourth and final attempt, we extend what is part of the context and introduce sessions. We combine two ideas:

1. 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. We therefore make an instance of the ideal world *stateful*, with the sequence of messages exchanged so far acting as associated data.
2. The sequence of messages exchanged so far provides uniqueness, as it grows at each call to wrap or unwrap, and it can therefore take over the role of the associated data A.

A *session* is the process in which the context grows with the sequence of messages exchanged so far. The object has become stateful and we include in the context the session history: the object identifier and the messages exchanged so far. In other words, the history is a sequence with ID followed by zero, one or more messages (A, P).

We specify the jammin cipher is Algorithm 1 in Section 2. Compared to Algorithm 8, we note the following differences:

- An instance was previously essentially static, with its attributes determined upon construction. Now the attribute inst.history dynamically evolves at each successful call to wrap or unwrap. A cryptogram can only be unwrapped successfully if all previous messages have been recovered correctly. An important application of this are *intermediate tags*.
- The codebook global variable now maps sequences of the form $(\mathrm{ID}; (A, P)^+)$ to cryptograms. Instances with equal ID but different inst.history attributes therefore produce independent cryptograms even for equal A values.
- The taboo global variable now maps sequences of the form $(\mathrm{ID}; (A, P)^*; A)$ to sets of cryptograms.
- The object can be cloned. The intention is to be able to simulate the real world's equivalent of making a copy of the state of the cipher. This means the adversary can, for instance, save the context and restart from it ad libitum.

B Deferred proofs

B.1 Proof of Theorem 1, the jammin cipher in OAE2

We start with proving a property of the jammin cipher when the encryption context is a nonce. In that case, the jammin cipher only excludes invalid cryptograms when generating a new cryptogram.

Proposition 5. *If the encryption context is a nonce and* WrapExpand *is t-expanding, on line 10 of Algorithm 1, we have that* $\mathcal{C} = \mathbb{Z}_2^{\mathrm{WrapExpand}(|P|)} \setminus \mathsf{taboo}(\mathsf{context})$.

Proof. We will show that, in these circumstances, we have

$$\mathsf{codebook}(\mathsf{context}; *) \cap \mathbb{Z}_2^{\mathrm{WrapExpand}(|P|)} = \varnothing.$$

As line 10 is under the "if", we know that $\mathsf{codebook}(\mathsf{context}; P) = \varnothing$. We split the proof in two cases depending on P.

- If $P \neq \epsilon$, and the encryption context is a nonce, we have that $\forall P' \neq \epsilon$, $\mathsf{codebook}(\mathsf{context}; P') = \varnothing$. It may happen that $\mathsf{codebook}(\mathsf{context}; \epsilon) \neq \varnothing$, but the condition on WrapExpand means that this set can only contain a string that is strictly shorter than WrapExpand($|P|$) bits.
- If $P = \epsilon$, we have that $\mathsf{codebook}(\mathsf{context}; \epsilon) = \varnothing$, but it may happen that $\mathsf{codebook}(\mathsf{context}; P') \neq \varnothing$ for some $P' \neq \epsilon$. However, the condition on WrapExpand means that this set can only contain a string that is strictly longer than WrapExpand(0) bits.

□

We continue with the proof of Theorem 1 itself.

Proof. We first describe how the jammin cipher is instantiated in the OAE2c security notion, then we tackle the bounds for privacy and authenticity.

Instantiation We define the $\mathcal{E}$ and $\mathcal{D}$ oracles in Real2C and Forge2C. Here, Real2C is a system with the same interface as Rand2C that is based on the jammin cipher. For the initialization, $S \leftarrow \mathcal{E}.\mathrm{init}(K, N)$ is instantiated as

$$S \leftarrow \mathcal{J}^{+t}.\mathsf{init}(0) \quad \text{followed by} \quad S.\mathsf{wrap}(N, \epsilon), \tag{2}$$

where the ID is arbitrarily set to 0 as the OAE2c security deals with only one key. To distinguish between the next and the last calls, we append a bit $\mathtt{0}$ or $\mathtt{1}$ to the associated data, hence $\mathcal{E}.\mathrm{next}(S, A, M)$ and $\mathcal{E}.\mathrm{last}(S, A, M)$ become $S.\mathsf{wrap}(A||\mathtt{0}, M)$ and $S.\mathsf{wrap}(A||\mathtt{1}, M)$, respectively. The initialization of $\mathcal{D}$ exactly like for $\mathcal{E}$ using Eq. (2), and $\mathcal{D}.\mathrm{next}(S, A, C)$ and $\mathcal{D}.\mathrm{last}(S, A, C)$ become $S.\mathsf{unwrap}(A||\mathtt{0}, C)$ and $S.\mathsf{unwrap}(A||\mathtt{1}, C)$, respectively.

Privacy When distinguishing between Real2C and Rand2C, the only difference is that Rand2C generates ciphertexts randomly with replacement, while the jammin cipher avoids ciphertexts already produced for the same associated data in the same context. Cryptogram collisions between plaintexts of equal length and same associated data and context are therefore the only way to distinguish the real from the ideal world. To quantify the collision probability in case of Rand2C, we can partition the query set it in subsets with equal context, associated data and length. If we denote the number of queries in subset i by q_i and the length of the plaintexts in this set by p_i, we obtain the following upper bound for the collision probability:

$$\Pr(\text{col}) \leq \sum_i \binom{q_i}{2} / 2^{p_i+t} = \sum_i \frac{q_i(q_i-1)}{2^{p_i+t+1}} .$$

As for some given length p there are only 2^p plaintexts, we have $q_i \leq 2^{p_i}$. Filling this in yields

$$\Pr(\text{col}) \leq \sum_i \frac{q_i - 1}{2^{t+1}} = \frac{\sum_i (q_i - 1)}{2^{t+1}} \leq \frac{q-1}{2^{t+1}} .$$

This bound is tight: the adversary can have all queries in one subset and take $q = 2^p$.

If the encryption context is a nonce, this strategy cannot be applied. Since Real2C never calls unwrap, taboo remains empty. Together with Proposition 5, this shows that the ciphertexts are always chosen from the complete set $\mathbb{Z}_2^{|P|+t}$ just like Rand2C, hence $\mathbf{Adv}^{\text{oae2-priv}}_{\mathcal{J}^{+t}}(\mathcal{D}) = 0$ in this case.

Authenticity In the Forge2C game, the adversary wins if it presents a cryptogram that was not previously generated. The jammin cipher always returns an error, hence the adversary never wins, and $\mathbf{Adv}^{\text{oae2-auth}}_{\mathcal{J}^{+t}}(\mathcal{D}) = 0$. □

B.2 Proof of Theorem 3, on the security of Deck-BO

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

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

where $\mathcal{D}''$ has the same resources as $\mathcal{D}$.

We then use Lemma 1 with $\mathcal{O} = \mathcal{J}^{+t} \triangleq \mathcal{J}$ and $\mathcal{P} = \text{Deck-BO}(\mathcal{RO}, t)$. In this proof, we use the session syntax of the jammin cipher. This is w.l.o.g. as in Deck-BO the history is coded as an injective function of the (A, P) pairs.

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

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

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. We use this to have a simple definition of forgery, namely as a successful unwrap record.

We have two types of bad event. If a transcript has a bad event, it is in $\mathcal{T}_{\text{bad}}$.

Tag collision : the transcripts has two records of the form $(\mathsf{wrap}, \mathsf{context}, P, C)$ and $(\mathsf{wrap}, \mathsf{context}, P', C')$ with $\epsilon \neq P \neq P' \neq \epsilon$ and the first t bits of C and C' are equal. Since $P \neq P'$, these first t bits are randomly generated with independent invocations of the random oracle, and we look at the probability of collisions among at most $\sigma(\mathsf{context})$ records.

Successful forgery : the transcript has a record $(\mathsf{unwrap}, \mathsf{history}, A, P, C)$ 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}.

This yields $\Pr\left(D_{\mathcal{P}} \in \mathcal{T}_{\text{bad}}\right) \leq \frac{q_{\mathsf{unwrap}}}{2^t} + \sum_{\mathsf{context}} \frac{\binom{\sigma(\mathsf{context})}{2}}{2^t}$.

We now argue that, for all $\tau \in \mathcal{T}_{\text{good}}$, we have $\Pr\left(D_{\mathcal{J}} = \tau\right) \geq \Pr\left(D_{\mathcal{P}} = \tau\right)$, 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 $\mathsf{context}$ 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 $\mathsf{context}$ value. In such records, there can be several records of the forms $(\mathsf{wrap}, \mathsf{context}, P, C)$ and $(\mathsf{unwrap}, \mathsf{context}, \perp, C)$.

In the jammin cipher, the value of each C in the wrap queries is taken from a set of cardinality at most $2^{|C|}$. Hence, each such record contributes a factor at least $2^{-|C|}$ to $\Pr\left(D_{\mathcal{J}} = \tau\right)$, conditioned on the previously considered records. Conditioned on the wrap records, the unwrap records contribute a factor 1 because the jammin cipher avoids all forgeries. The jammin cipher may return an error, but thanks to Proposition 2, this would require $q_{\mathsf{unwrap}} \geq 2^t$.

In $\mathcal{P} = \text{Deck-BO}(\mathcal{RO}, t)$, the value of each C in the wrap queries is obtained from $T = \mathcal{RO}_1(\mathsf{context}; P)$ for the first t bits and $P + \mathcal{RO}_2(\mathsf{context}; T)$ for the last bits, with the subscripts indicating domain separation. Given that all the tags T are distinct in $\mathcal{T}_{\text{good}}$, the probability of each wrap record contributes a factor exactly $2^{-|C|}$ to $\Pr\left(D_{\mathcal{P}} = \tau\right)$. Conditioned on that, an unwrap record naturally contributes at most 1 to $\Pr\left(D_{\mathcal{P}} = \tau\right)$.

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

B.3 Proof of Theorem 4, on the security of Deck-BOREE

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

$$\begin{aligned}&\Delta_{\mathcal{D}}(\text{Deck-BOREE}(F,t)\ ;\ \mathcal{J}^{+t})\\ &\qquad\qquad \leq \Delta_{\mathcal{D}''}(\text{Deck-BOREE}(\mathcal{RO},t)\ ;\ \mathcal{J}^{+t}) + \mathbf{Adv}_F^{\text{prf}}(\mathcal{D}'),\end{aligned}$$

where $\mathcal{D}''$ has the same resources as $\mathcal{D}$.

We then use Lemma 1 with $\mathcal{O} = \mathcal{J}^{+t} \triangleq \mathcal{J}$ and $\mathcal{P} = \text{Deck-BOREE}(\mathcal{RO}, t)$. In this proof, we use the session syntax of the jammin cipher. This is w.l.o.g. as in Deck-BOREE the history is coded as an injective function of the (A, P) pairs.

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

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

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. We use this to have a simple definition of forgery, namely as a successful unwrap record.

There are additional components in the records.

1. hidden diversifier V, present in both wrap and unwrap records. It is there to define bad events and disclosed only at the end of the game.
   - In Deck-BOREE, upon wrapping, this is the tag before it is blinded. It can be expressed from the ciphertext $C = W||Z$ with $|W| = t$ and $|Z| = |C| - t$, see Figure 2, as $V = W + \mathcal{RO}(\mathsf{context}; Z||\mathsf{111})$
   - In the jammin cipher, V is randomly generated from the ciphertext, but in a consistent way: $V = W + \mathcal{RO}_3(\mathsf{context}; Z)$, with $\mathcal{RO}_3$ being statistically independent of the coins used in the jammin cipher.
2. Unverified decrypted ciphertext Y in unwrap records. It may be disclosed upon each unwrap query so that the adversary can react adaptively. For each query, the adversary decides whether she accesses the unverified decrypted ciphertext; if not, $Y = \perp$.
   - In Deck-BOREE, $Y = Z + \mathcal{RO}(\mathsf{context}; V||\mathsf{101})$. In the case of a valid unwrap query, Y would be the actual plaintext.
   - In the jammin cipher, Y is generated randomly and independently of the coins it uses internally as $Y = \mathsf{0}^{|C|-t} + \mathcal{RO}_2(\mathsf{context}; C)$, so Y bears no relation with any plaintext.

We have two types of bad event. If a transcript has a bad event, it is in $\mathcal{T}_{\text{bad}}$.

Hidden tag (V) collision : the transcript has two distinct records of the form $(\mathsf{wrap}, \mathsf{context}, P, C, V)$ or $(\mathsf{unwrap}, \mathsf{context}, P, C, V, Y)$ with equal $\mathsf{context}$, $|C| > t$, $Y \neq \perp$ (if unwrap) and equal values V. The probability of collision among wrap queries is as in Theorem 3. For unwrap queries, the probability is similar, with two cautionary notes: 1) the values of V cannot be used adaptively to adjust W and obtain a collision and 2) a strategy where the adversary varies W but not Z is pointless as the resulting V cannot collide. Finally, collisions between wrap and unwrap queries must also be taken into account, hence we get a total of $\binom{\sigma'(\mathsf{context})}{2}$ pairs to consider.

Successful forgery : the transcript has a record $(\mathsf{unwrap}, \mathsf{history}, A, P, C, V, Y)$ with $P \neq \perp$. A forgery cannot happen in the jammin cipher, but it can in Deck-BOREE.

This yields $\Pr(D_{\mathcal{P}} \in \mathcal{T}_{\text{bad}}) \leq \frac{q_{\mathsf{unwrap}}}{2^t} + \sum_{\mathsf{context}} \frac{\binom{\sigma'(\mathsf{context})}{2}}{2^t}$.

We now argue 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 $\mathsf{context}$ 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 $\mathsf{context}$ value. In such records, there can be several records of the forms $(\mathsf{wrap}, \mathsf{context}, P, C, V)$ and $(\mathsf{unwrap}, \mathsf{context}, \perp, W||Z, V, Y)$.

In the jammin cipher, the value of each C in the wrap queries is taken from a set of cardinality at most $2^{|C|}$. Hence, each such record contributes a factor at least $2^{-|C|}$ to $\Pr(D_{\mathcal{J}} = \tau)$, conditioned on the previously considered records. Conditioned on the wrap records, the unwrap records contribute only through their values V and Y (when $Y \neq \perp$) because the jammin cipher avoids all forgeries. As $Y = 0^{|Z|} + \mathcal{RO}_2(\mathsf{context}; W||Z)$, this contributes a factor $2^{-|Z|}$ exactly. As $V = W + \mathcal{RO}_3(\mathsf{context}; Z)$, queries with a fresh Z contribute 2^{-t}, while those that vary only on W contribute 1 conditioned on the first query with Z. The jammin cipher may return an error, but thanks to Proposition 2, this would require $q_{\mathsf{unwrap}} \geq 2^t$.

In $\mathcal{P}$ = Deck-BOREE$(\mathcal{RO}, t)$, the value of each $C = W||Z$ in the wrap queries is obtained as $Z = P + \mathcal{RO}_2(\mathsf{context}; V)$ and $W = 0^t + \mathcal{RO}_1(\mathsf{context}; P) + \mathcal{RO}_3(\mathsf{context}; Z)$. Given that all the P values and V values are distinct in $\mathcal{T}_{\text{good}}$, the contribution of C is a factor exactly $2^{-|C|}$ to $\Pr(D_{\mathcal{P}} = \tau)$. Conditioned on the wrap records, we look at the contributions of the values of V and Y (when $Y \neq \perp$) in the unwrap records. The contribution of $P = \perp$ amounts to at most 1 in $\Pr(D_{\mathcal{P}} = \tau)$ as forgeries may happen. As $Y = Z + \mathcal{RO}_2(\mathsf{context}; V)$ and the V values in the unwrap records are distinct (also distinct from the V values in the wrap records), the value of Y of an unwrap record contributes a factor $2^{-|Z|}$ exactly. For wrap queries, $V = 0^t + \mathcal{RO}_1(\mathsf{context}; P)$, which contributes a factor exactly 2^{-t} since all the P values are distinct. For unwrap queries, $V = W + \mathcal{RO}_3(\mathsf{context}; Z)$ and queries with a fresh Z contribute 2^{-t}, while those that vary only on W contribute 1 conditioned on the first query with Z.

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

B.4 Proof of Theorem 5, on the security of Deck-JAMBO

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

$$\Delta_{\mathcal{D}}(\text{Deck-JAMBO}(F, t, \text{expand}, \text{split}) \ ; \ \mathcal{J}) \\ \leq \Delta_{\mathcal{D}''}(\text{Deck-JAMBO}(\mathcal{RO}, t, \text{expand}, \text{split}) \ ; \ \mathcal{J}) + \mathbf{Adv}_F^{\text{prf}}(\mathcal{D}'),$$

where $\mathcal{D}''$ has the same resources as $\mathcal{D}$.

We then use Lemma 1 with $\mathcal{O} = \mathcal{J}$ and $\mathcal{P} = \text{Deck-JAMBO}(\mathcal{RO}, \dots)$.

In this proof, we use the session syntax of the jammin cipher. However, in Deck-JAMBO, what is kept in story is not the plaintext but the *plaintext representative* Y computed as detailed shortly. Below, we include collisions in Y as bad events. If no bad events happen, story is an injective function of the (A, P) pairs.

The plaintext representative Y is computed as follows. Let $(L_0, L_+, R_0, R_+) = \text{split}(\text{expand}(P, t), t)$. Then,

$$Y = \left(P^{\text{last}}||\mathtt{10}^{\text{pad}}\right) + \left(\mathcal{RO}(\text{story}; A||\mathtt{10}; \mathtt{0}^t||P^{\text{first}}||\mathtt{001})||\mathtt{0}^{r_+}\right) \tag{3}$$

with $L_0||L_+ = \mathtt{0}^t||P^{\text{first}}$, $R_0||R_+ = P^{\text{last}}||\mathtt{10}^{\text{pad}}$, $l = |L_0| + |L_+|$, $r_0 = |R_0|$ and $r_+ = |R_+|$.

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

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

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. We use this to have a simple definition of forgery, namely as a successful unwrap record.

There is an additional component in the wrap records, namely the plaintext representative Y. The value of Y is used to define bad events, and it is disclosed only at the end of the game. In all cases, $Y = \epsilon$ when $P = \epsilon$.

- In Deck-JAMBO, it is computed as in Equation (3). I it is the right branch of B after the first round, as can be seen in Figure 2.
- In the jammin cipher, Y is randomly generated, but in a consistent way, namely, $Y = (P^{\text{last}}||\mathtt{10}^{\text{pad}}) + (\mathcal{RO}_0(\mathsf{context}; P^{\text{first}})||\mathtt{0}^{r_+})$, with $\mathcal{RO}_0$ being statistically independent of the coins used in the jammin cipher.

We have two types of bad event. If a transcript has a bad event, it is in $\mathcal{T}_{\text{bad}}$.

Plaintext representative (Y) collision The transcript has two records of the form $(\mathsf{wrap}, \mathsf{context}, P, C, Y)$ and $(\mathsf{wrap}, \mathsf{context}, P', C', Y')$ with $\epsilon \neq P \neq P' \neq \epsilon$ and $Y = Y'$. In Deck-JAMBO, the first r_0 bits of Y are randomly generated with $\mathcal{RO}$. As Y is disclosed only at the end of the game, the value of P^{last} cannot be used adaptively to compensate for changes in $\mathcal{RO}(\dots; P^{\text{first}}||\dots)$ and obtain a collision. Also, a strategy where the adversary varies P^{last} but not P^{first} is pointless as the resulting Y cannot collide. In the end, we look at the probability of collisions on $r_0 \geq 2t$ bits among at most $\sigma(\mathsf{context})$ records.

Left branch (V) collision (collision in the first l bits of C) : The transcript has two records of the form $(\mathsf{wrap}, \mathsf{context}, P, C, Y)$ and $(\mathsf{wrap}, \mathsf{context}, P', C', Y')$ with $\epsilon \neq P \neq P' \neq \epsilon$ and the first l bits of C and C' are equal. Assuming $Y \neq Y'$, these first l bits are randomly generated with independent invocations of the random oracle, and we look at the probability of collisions on $l \geq 2t$ bits among at most $\sigma(\mathsf{context})$ records.

Successful forgery : The transcript contains a record $(\mathsf{unwrap}, \mathsf{context}, P, C)$ with $P \neq \perp$. A forgery cannot happen in the jammin cipher, but it can in Deck-JAMBO. After q_{unwrap} queries, the probability that the left branch starts with t zeroes for one of the queries is $q_{\mathsf{unwrap}}2^{-t}$. This is valid even if there are collisions at the input of the middle round.

Hence $\Pr\left(D_{\mathcal{P}} \in \mathcal{T}_{\text{bad}}\right) \leq \frac{q_{\mathsf{unwrap}}}{2^t} + \sum_{\mathsf{context}} \frac{\binom{\sigma(\mathsf{context})}{2}}{2^{2t-1}}$.

We now argue that, for all $\tau \in \mathcal{T}_{\text{good}}$, we have $\Pr\left(D_{\mathcal{J}} = \tau\right) \geq \Pr\left(D_{\mathcal{P}} = \tau\right)$, 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 $\mathsf{context}$ 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 $\mathsf{context}$ value. In such records, there can be several records of the forms $(\mathsf{wrap}, \mathsf{context}, P, C, Y)$ and $(\mathsf{unwrap}, \mathsf{context}, \perp, C)$.

In the jammin cipher, each wrap record with $P \neq \epsilon$ and a new value of P^{first} contributes a factor 2^{-r_0} due to the value of Y. The value of each C in the wrap queries is taken from a set of cardinality at most $2^{|C|}$. Hence, each such record contributes a factor at least $2^{-|C|}$ to $\Pr\left(D_{\mathcal{J}} = \tau\right)$, conditioned on the previously considered records. Conditioned on the wrap records, the unwrap records contribute a factor 1 because the jammin cipher avoids all forgeries. The jammin cipher may return an error, but thanks to Proposition 2, this would require $q_{\mathsf{unwrap}} \geq 2^t$.

In $\mathcal{P} = $ Deck-JAMBO$(\mathcal{RO}, \dots)$, each wrap record with $P \neq \epsilon$ and a new value of P^{first} contributes a factor 2^{-r_0} due to the value of Y. The value of each C in the wrap queries is obtained as $V = \mathtt{0}^t||P^{\text{first}} + \mathcal{RO}(\mathsf{context}; Y||\mathtt{011})$ for the first l bits and $P^{\text{last}}||\mathtt{10}^{\text{pad}} + \mathcal{RO}(\mathsf{context}; V||\mathtt{101})$ for the last bits. Given that all the Y values and V values are distinct in $\mathcal{T}_{\text{good}}$, the probability of each wrap record contributes a factor exactly $2^{-|C|}$ to $\Pr\left(D_{\mathcal{P}} = \tau\right)$. Conditioned on that, an unwrap record contributes at most 1 to $\Pr\left(D_{\mathcal{P}} = \tau\right)$.

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

C OAE2c security

In figure [15, Fig. 6], the privacy and authentication games used for defining OAE2c security are described. In contrast to the OAE2a and OAE2b security definitions, privacy and authentication are defined separately. Concretely, the privacy advantage is given by the adversary's ability to distinguish between the games Real2C and Rand2C, where Rand2C represents the ideal world and Real2C represents the real world that uses a concrete cipher. Also, note that no decryption oracle is given in the ideal world but that it only returns the expected number of uniformly random bits. Then, regarding the authentication advantage, no ideal world is queried. The advantage is defined using the game Forge2C, that is, Real2C including the finalize procedure, as the probability that an adversary produces an input that, when presented to the finalize procedure, the procedure will evaluate it to true. Combining these two advantages, the authors of [15] conclude that OAE2c security is achieved if an adversary has a small privacy advantage as well as a small authentication advantage.

```
proc initialize                                   $\boxed{\textbf{Real2C}_\Pi}$  $\boxed{\textbf{Forge2C}_\Pi \leftarrow}$
$I \leftarrow 0$;  $K \twoheadleftarrow \mathcal{K}$
$\mathcal{Z} \leftarrow \emptyset$

proc Enc.init($N$)
if $N \notin \mathcal{N}$ then return $\perp$
$I \leftarrow I+1$;  $S_I \leftarrow \mathcal{E}$.init$(K, N)$
$N_I \leftarrow N$;  $\boldsymbol{A}_I \leftarrow \boldsymbol{M}_I \leftarrow \boldsymbol{C}_I \leftarrow \Lambda$
return  $I$

proc Enc.next($i, A, M$)
if $i \notin [1..I]$ or $S_i = \perp$ then return $\perp$
$(C, S_i) \leftarrow \mathcal{E}$.next$(S_i, A, M)$
$\boldsymbol{A}_i \leftarrow \boldsymbol{A}_i \parallel A$;  $\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i \parallel M$;  $\boldsymbol{C}_i \leftarrow \boldsymbol{C}_i \parallel C$
$\mathcal{Z} \leftarrow \mathcal{Z} \cup \{(N_i, \boldsymbol{A}_i, \boldsymbol{C}_i, 0)\}$
return $C$

proc Enc.last($i, A, M$)
if $i \notin [1..I]$ or $S_i = \perp$ then return $\perp$
$C \leftarrow \mathcal{E}$.last$(S_i, A, M)$;  $S_i \leftarrow \perp$
$\boldsymbol{A}_i \leftarrow \boldsymbol{A}_i \parallel A$;  $\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i \parallel M$;  $\boldsymbol{C}_i \leftarrow \boldsymbol{C}_i \parallel C$
$\mathcal{Z} \leftarrow \mathcal{Z} \cup \{(N_i, \boldsymbol{A}_i, \boldsymbol{C}_i, 1)\}$
return $C$

proc finalize $(N, \boldsymbol{A}, \boldsymbol{C}, b)$                                   $\leftarrow$
if $|\boldsymbol{A}| \neq |\boldsymbol{C}|$ or $|\boldsymbol{A}| = 0$ or $(N, \boldsymbol{A}, \boldsymbol{C}, b) \in \mathcal{Z}$ then return false
$S \leftarrow \mathcal{D}$.init$(K, N)$;  $m \leftarrow |\boldsymbol{C}|$                              $\leftarrow$
for $i \leftarrow 1$ to $m - b$ do                                         $\leftarrow$
    $(M, S) \leftarrow \mathcal{D}$.next$(S, \boldsymbol{A}[i], \boldsymbol{C}[i])$                      $\leftarrow$
    if $M = \perp$ then return false                               $\leftarrow$
if $b = 1$ and $\mathcal{D}$.last$(S, \boldsymbol{A}[m], \boldsymbol{C}[m]) = \perp$ then return false
return true                                                    $\leftarrow$
```

```
proc initialize                                   $\boxed{\textbf{Rand2C}_\Pi}$
$I \leftarrow 0$
$E(x) \leftarrow$ undef for all $x$

proc Enc.init($N$)
if $N \notin \mathcal{N}$ then return $\perp$
$I \leftarrow I+1$
$N_I \leftarrow N$;  $\boldsymbol{A}_i \leftarrow \boldsymbol{M}_i \leftarrow \Lambda$
return  $I$

proc Enc.next($i, A, M$)
if $i \notin [1..I]$ or $N_i = \perp$ then return $\perp$
$\boldsymbol{A}_i \leftarrow \boldsymbol{A}_i \parallel A$;  $\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i \parallel M$
if $E(N_i, \boldsymbol{A}_i, \boldsymbol{M}_i, 0) =$ undef then
    $E(N_i, \boldsymbol{A}_i, \boldsymbol{M}_i, 0) \twoheadleftarrow \{0,1\}^{|M|+\tau}$
$C \leftarrow E(N_i, \boldsymbol{A}_i, \boldsymbol{M}_i, 0)$
return $C$

proc Enc.last($i, A, M$)
if $i \notin [1..I]$ or $N_i = \perp$ then return $\perp$
$\boldsymbol{A}_i \leftarrow \boldsymbol{A}_i \parallel A$;  $\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i \parallel M$
if $E(N_i, \boldsymbol{A}_i, \boldsymbol{M}_i, 1) =$ undef then
    $E(N_i, \boldsymbol{A}_i, \boldsymbol{M}_i, 1) \twoheadleftarrow \{0,1\}^{|M|+\tau}$
$C \leftarrow E(N_i, \boldsymbol{A}_i, \boldsymbol{M}_i, 1)$;  $N_i \leftarrow \perp$
return $C$
```

Fig. 3: Overview of OAE2c as defined in [15].