Decryption Despite Errors

Achieving security against fuzzing and improving bandwidth efficiency by combining FEC and symmetric encryption.

Karolin Varner karo@cupdev.net

DRAFT GENERATED 2021-06-20

Abstract

This paper introduces the notion of decryption despite errors: ciphers with forward error correction properties that provide specific security properties in relation to noise or adversarial error in the ciphertext. The two properties are security against fuzzing and partial message recovery. The latter refers to a ciphers ability to decrypt ciphertexts even if the noise level is too high to recover the original message. Definitions of and fundamental limits of security under these constraints are presented. Security against fuzzing is formalized. PMR is incompatible with CCA2 security, but can still provide security: Informally, it should be infeasible for an attacker to do any better than raising the (random) noise level in the plain text in the attempt to forge messages. Multiple PMR security notions are formalized; the most interesting one states, that asymptotic infeasibility of distinguishing the decryption of two challenger-chosen error-patterns (bit flips) with the same number of errors must be demonstrated for security.

The outline of a construction that operates as a mode for a pseudo random function is presented; the construction is based on multiple rounds of an error correcting code and interleaver, both randomized using the PRF generated key stream.

A novel, fast, bitwise shuffle for cryptographic purposes is introduced; the algorithm transforms a random oracle into a shuffle by splitting the input into blocks. Blocks are mixed with `swap_by_mask` and a random mask; the relative position of bits are decorrelated using bitwise rotation.

This paper is currently a draft and likely to contain a number of errors. MUMBLE MUMBLE indicates references and content to be filled in. Proofs of security and correctness are to be done. For now, the constructions have informal arguments of security/correctness.

1 Quick Reference

1.1 Standard operations

$\cdot \oplus \cdot$	Exclusive or over bits or vectors of bits.
$W(\cdot)$	The hamming weight (number of bits set to one in a vector of bits).
$W(\cdot \oplus \cdot)$	The hamming distance.
$\lvert \cdot \rvert$	Size of a vector.
$mode(\cdot)$	Mode of some statistical distribution.

1.2 Decryption Despite Errors

Schemes are said to have security against fuzzing (`SAF`) if they can be shown to support `FEC-ATK` security. Schemes are said to support Partial Message Recovery (`PMR`) if they can securely decrypt when `w >`

0 (i.e. there will be errors in the decrypted plain text). `proportional-loss-` and `loss-estimate-` security notions apply. Decryption Despite Errors (`DDE`) refers to schemes that support both `SAF` and `PMR`; the associated security notion is `DDE-ATK`. This paper omits the study of scheme that supports `PMR` but not `SAF`.

The definition of a symmetric cipher has been adapted for this paper to be applicable to partial message recovery. A redundancy parameter is added; the scheme takes the ciphertext or a cipher-text with errors and returns the original plaintext or a related plaintext as well as an error estimate. The error estimate w replaces the usual construction in authenticated encryption, where a message or the bottom element is returned. The message is rejected if $w > w_{max}$.

The paper also discusses the case of standard authenticated definitions with additional security against fuzzing. In this case, standard definitions are applicable.

`DDE`	"Decryption Despite Errors".
`PMR`	"Partial Message Recovery".
`SAF`	"Security Against Fuzzing".
$Enc_{K,R}(k,n,x) = y$	The polynomial time encryption algorithm.
	The shorthand notations $Enc(k,n,x)$ and $Enc(x)$ may be used.
$Dec_{K,R}(k,n,y') = (x',w)$	The polynomial time decryption algorithm.
	The shorthand notations $Dec(k,n,y')$ and $Dec(y')$ may be used.
K	The security parameter.
R	Redundancy parameter.
$w_{max} : \mathbb{Q}$	"Maximum allowed loss".
	If $w > w_{max}$, Dec will discard the message.
	If $w_{max} = 0$, the loss estimate shall be a normal message authentication code.
$(k,n) : \{0,1\}^*$	Symmetric key, nonce.
$(x,y) : \{0,1\}^*$	"Original" plaintext/ciphertext. (Without errors).
$(x',y') : \{0,1\}^*$	"Derived" plaintext/ciphertext. (With errors).
$s = y \oplus y'$	"The syndrome". Error vector in the ciphertext.
$r = x \oplus x'$	"The residual". Error vector after decryption in the plaintext.
$W_s = W(s) = W(y \oplus y')$	"The syndrome weight". Number of bit flips in the ciphertext.
$W_r = W(r) = W(x \oplus x')$	"The residual weight". Number of bit flips in the plaintext.
$w : \mathbb{Q}, w \approx W_r$	"Loss estimate". Generalization of message authentication to soft decision.

1.3 Security Notions & Instantiations

`AESAF` and `DDE-ATK` are composite notions intended for use in actual constructions; they constructions are called `dde-cca1-cipher`, `dde-cca2-cipher` and `aefec-cipher`. `dee-cca2-cipher-nopunct` is a special variant. The others are the basic notions the composite notions are made up.

The attack `ATK` is used throghout the paper. `ATK` $\in$ {`CCA1`, `CCA2`}.

`AESAF` $\iff$ `IND-CCA2` $\wedge$ `FEC-CCA2`.
`DDE-ATK` $\iff$ `IND-CCA1` $\wedge$ `NM-CCA1` $\wedge$ `pl-IND-ATK` $\wedge$ `pl-NM-ATK` $\wedge$ `LEU-ATK` $\wedge$ `FEC-ATK`.

`AESAF`	"Authenticated encryption with security against fuzzing".
`DDE-ATK`	"Decryption despite errors".
`FEC-ATK`	"Forward error correction non malleability".
	Security against fuzzing as a stand alone notion.
`pl-NM-ATK`	"Non-malleability up to proportional loss".
`pl-IND-ATK`	"Decryption-indistinguishably up to proportional loss".

`LEU-ATK`	"Loss-estimate unforgability".

2 Introduction

2.1 Use Case

All physical channels are nosy, meaning that the rate of transmission errors can be reduced, but never to zero. Reducing the noise level to the degree required for most applications by increasing the reliability of the physical system is often impractical. There is a better way to deal with this: Forward Error Correction — codes designed to detect and correct errors in return for extra overhead during transmission (hence the "forward"). These codes often make some assumptions about the distribution of errors in the channel and yield reduced error correction rates when these assumptions fail to be met. Some codes operate at much greater efficiencies when the errors are randomly distributed, while others are most efficient against bursts . Convolutional codes for instance suffer from error floor problems. Interleaving schemes are used to compose such codes, yielding combined that correct errors reliably in a greater range of scenarios. For instance, CDs use a complex scheme of interleaved Reed-Solomon codes to correct errors.

Cryptographic systems are tend to be layered on top of these error corrected channels . With overwhelming probability, data protected with authenticated encryption either arrives as sent by the communication partner or it does not arrive at all. Authenticated encryption guarantees authenticity, integrity and confidentiality of the data transmitted but not reliability of the transmission. In fact, AE makes denial of service attacks significantly easier because it ensures as a property that a minimum number of modifications to the ciphertext suffices to ensure the message is rejected. One bit flip, erasure, or insertion enough to have the message rejected. This is a necessity to achieve security under an adaptive chosen ciphertext attack (`CCA2`) [citation needed] — the common definition provided by most security protocols today.

The forward error correction employed on communication channels improve that situation, but being designed for specific noise profiles, finding some error pattern that has the message rejected is often easy for a determined adversary. The practical implications of that situation are limited: fuzzing of radio transmissions can usually be addressed by shutting of the fuzzer; most transmissions are secured physically, and data rates are large enough that a few packets lost do not make a big difference.

Still, improving the reliability of error correction in data transmissions has significant potential to improve transmission efficiencies and reliability, especially in niche use cases like long range radio transmission or media streaming. Common use cases such as mobile networks do not require efficiency improvements for functionally, but even very slight efficiency improvements have the potential for great economic impact du to the sheer scale of deployment.

2.2 Existing work

2.2.1 Approximate Message Authentication

There is some work on the problem of generalizing MACs to messages that are *similar* to the original. Such schemes are called *Approximate* Message Authentication Codes; see MUMBLE MUMBLE [citation needed] for a relatively recent work with good references. While a valid MAC implies with overwhelming probability that the tag was generated from the exactly same message as is being verified, AMACs are valid if the derived message x' is within some distance $d : D$ of the original x where D is some metric. d is usually some constant; i.e. AMACs output a hard decision. For the schemes presented in this paper, more precise information about the distance between x and x' would be preferable.

That is the scheme should return d instead of just outputting a decision $D(x, x') \geq d$.

2.2.2 Existing hybrid FEC and symmetric crypto schemes

There has also been some work on producing hybrid ciphers integrating error correction and block ciphers; MUMBLE MUMBLE [citation needed] produced what they call a high diffusion cipher. I know of no attacks against this work, but earlier constructions employing FEC to achieve security have been proven insecure. The work referenced here is a from scratch construction of a PRP, and therefore unfortunately hard to prove secure. This is not usual in PRPs and also the case for very widely used constructions [citation needed], however since this is a fairly niche application cryptanalysis is likely to focus on one of these more widely used constructions. Confidence in this specific scheme is therefore limited.

2.3 Contributions in this paper

This paper defines two basic properties symmetric ciphers may have and (plans to in the future when this is no longer a draft) provide formal security definitions modeling both properties as well as analysis of the maximum achievable security for each property. `PMR` is incompatible with full `IND/NM-CCA2` security, so a formalization of `CCA2` "up to" `PMR` is presented. `FEC` is compatible with `CCA2`.

Schemes conjectured to possess these properties are presented.

2.3.1 Security Against Fuzzing

Authenticated encryption schemes with this property ensure, that given a limited maximum number of bit flips, finding some error pattern in the ciphertext that maximizes the number of bit flips in the plaintext is computationally hard. Basically, this formalizes the security of the forward error correction and provides error correction even in the presence of a determined adversary trying to jam message transmission with while minimizing their effort. It also ensures that as long as some information can be transmitted, there exists some redundancy parameter for the cipher that will extract the message being transmitted successfully (although trying this in the real world might be entirely impractical).

2.3.2 Partial Message Recovery

Authenticated encryption schemes that allow the decryption of a ciphertext even if the original message cannot be recovered. Such schemes are by their very nature malleable schemes, analysis will focus on what standard notions of security can be achieved, what new notions of security can be introduced in this setting and how these relate to standard notions. The practical use cases for schemes with this property is increasing transmission efficiency if the data being transmitted has some inherent redundancy (e.g., media streaming).

The idea is this: A dropped package will generally result in big artifacts; a handful of bit flips on the other hand are barely noticeable in some types of media. Right now, the redundancy level has to be chosen such that a single bit flip occurs on average in a very small proportion of messages to avoid packet loss. With `PMR`, a much lower redundancy level could be chosen such that the effective error rate is closer to a few per packet because a few bit flips wold no longer be catastrophic.

3 Notions of security

3.1 Security Against Fuzzing: FEC Non-Malleability

Informally, a scheme is secure against fuzzing if it is hard for an adversary to thwart it's successful decoding by flipping bits. The number of bit flips is chosen as the resource limited to the adversary.

This restriction on fuzzer capability is necessary because if all information is erased from the message, recovering the message would obviously be impossible. This model is also justified by real-world application as it captures the scenario of a radio transmitter in a shared medium (e.g. wifi) whose goal it is to overwhelm the communications channel with noise. It appears likely that drastically reducing the transmission rate of the radio channel would be relatively easy for the adversary, while further decreases in transmission rate would be asymptotic (i.e. there are diminishing returns). Whether this model captures reality needs to be empirically established, but this is out of scope for this paper.

A proper security definition in the game playing framework by [citation needed] is planned. For now this needs to suffice: A scheme is considered to be `FEC-ATK` secure if an adversarie's advantage in winning the game is negligible.

1. Adversary produces a pair of messages x_0, x_1 of the same length and along with according redundancy parameters R_0, R_1. The adversary is given access to the relevant oracles.
2. The game encrypts the messages at the specified redundancy level, yielding y_0, y_1.
3. Adversary produces a pair of syndromes s_1, s_2 of the same weight (but not the same length) with access to the relevant oracles and state made in 1.
4. Game decrypts the derived messages at the specified redundancy parameters.
5. Adversary wins if the redundancy levels do not predict the residual weight, provided they did not try to cheat with the syndrome weight or message length.

$$comp(R_0, R_1) \neq comp(W_{r_0}, W_{r_1}) \wedge |x_0| = |x_1| \wedge W_{s_0} = W_{s_1}$$

$$comp(a, b) = \begin{cases} a < b : -1 \\ a = b : 0 \\ a > b : 1 \end{cases}$$

When applied to a `CCA2` secure authenticated encryption (i.e., a scheme without partial message recovery) scheme, set $w \leftarrow 0$ if the message passes authentication and $w \leftarrow 1$ otherwise.

3.2 Partial Message Recovery

`PMR` covers the notion that it should be possible to decrypt a message even if the syndrome weight is beyond the schemes' ability to recover the original message. If the original message cannot be recovered, then the decryption algorithm must return some other message in its place, so the scheme is necessarily cipher-text and plain-text malleable which has a direct, negative impact on security. The key insight here is that to be secure, an adversary should be be unable to do anything but raise the noise level; a channel with a dedicated adversary systematically flipping bits should look just like a channel with an increased noise level to the recipient.

More formally, for every syndrome space containing all syndromes of the same hamming weight, there is some probability space of residuals. This space must be the same for each element of the syndrome space.

Since the scheme returns an error estimate instead of a hard decision, a new unforgability notion applicable to the scheme has to be arrived at.

3.2.1 Incompatibility with NM-CPA

Partial Message Recovery is incompatible with `NM-CPA`, `NM-CCA1` and `NM-CCA2`. Since `IND-CCA2` $\iff$ `NM-CCA2`, `PMR` is also incompatible with `IND-CCA2`. To brake non-malleability games an adversary can simply flip a single bit in the cipher-text and then construct a relation from the hamming distance.

A scheme that cannot provide NM-CPA should usually not be considered secure enough for any real-world application beyond the construction of other, more secure, schemes. Let me point out though that the usual security definitions don't capture the properties of partial message recovery very well. PMR can provide a decent level of security, even under adaptive chosen ciphertext attack, if the security definitions are made aware of the malleability properties. This may seem like a potentially dangerous move, but there is some precedence: Ciphers generally do not hide the size of the message, so the usual security definitions were defined to not include attacks based on the size of the message. Analogously to the restriction of standard security notions to messages of the same length, the next sections will define further weakened security notions applicable to PMR.

3.2.2 pl-IND-ATK: indistinguishably up to proportional-loss

A proper game-based definition needs to be created still. For now, the following outline based on [citation needed] is given:

1. The adversary chooses a message x.
2. Game encrypts the state $y = Enc(x)$.
3. Adversary chooses two syndromes of the same hamming weight s_0, s_1 with access to the usual oracles and the state produced in 1.
4. Game randomly applies and decrypts one of the syndromes:

$$b =\leftarrow^R \{0,1\}; (x', w) = Dec(y \oplus s_b)$$

5. Adversary outputs a guess of b given x', the oracles, and the state from 3.
6. Challenger wins if their guess is right, and they didn't cheat with syndrome weight:

$$b = b_g \wedge W_{s_0} = W_{s_1}$$

The adversary also loses if they decrypted $y \oplus s_0$ or $y \oplus s_1$ using the oracle. Note that this definition is not the usual *ciphertext indistinguishability*; it is indistinguishibility of a derived plain text

3.2.3 pl-NM-ATK: non-malleability up to proportional-loss

Again, a proper game is yet to be arrived at. For now, the following outline based on [citation needed] is used:

1. The adversary chooses a valid message space M.
2. The game chooses two messages at random $x_0, x_1 \leftarrow^R M$.
3. The game encrypts both $y_0 \leftarrow Enc(x_0); y_1 \leftarrow Enc(x_1)$.
4. The adversary outputs a relation and a vector of syndromes of the same hamming weight (R, S) given y_1, oracles, and the state from 1.
5. The game chooses a random syndrome of the same hamming weight, applies it two both ciphertexts and decrypts them:

$$s \leftarrow^R \{0,1\}^{|S_0|}; x'_0 = Dec(y_0 \oplus s), x'_1 = Dec(y_1 \oplus s)$$

6. The game applies each syndrome and decrypts the resulting ciphertexts: $(X', W') \leftarrow Dec(y_1 \oplus S)$.
7. The game randomly chooses one of the two messages $b \leftarrow^R \{0,1\}$.
8. The adversary wins the game if the relation can determine the value of b, and they didn't attempt to cheat by outputting syndromes of different weights or of weight zero:

$$R(x'_b, E') = b \wedge (\forall s_u, s_v \in S, W(s_u) = W(s_v) \wedge W(s_u) > 0)$$

3.2.4 LEU-ATK: Loss estimate unforgeability

The loss estimate w is to PMR what the MAC is to authenticated encryption. Outline of the game:

1. The game takes the security parameter K and a parameter $\Delta w : \mathbb{Q}$; the encryption oracle keeps track of all encryptions in the encryption log L.
2. The adversary outputs a message x.
3. The game encrypts the message $y \leftarrow Enc(x)$.
4. The adversary is given y and outputs some derived ciphertext y' with access to the oracles, and the state it produced in 2.
5. The game decrypts the derived ciphertext $(x', w) = Dec(y')$.
6. The game looks up the closest original plaintext/ciphertext pair $(\hat{x}, \hat{y})$ produced with the encryption oracle such that
$$\forall (x_u, y_u) \in L, W(\hat{y} \oplus y') \leq W(y_u \oplus y')$$
7. The adversary wins if
$$|w - W(x \oplus \hat{x})| > \Delta w$$

A scheme is considered to be LEU-ATK secure if an adversaries probability of winning the game is negligible in $K * \Delta w$.

3.2.5 The Short-Distance Brute Force Attack

Weakening IND/NM-CCA2 security has a real impact; there are attacks that are possible against PMR schemes that are impossible against AE. Specifically, PMR offers very weak protection against an attacker producing specific, low weight residuals under a CCA2 attack. High weight residuals are not affected.

There exists some function $f(W_r) = W_s$ that assigns each residual weight some syndrome weight that maximizes the probability of producing a syndrome weighing W_r. This function is public. Practically, this function can be utilized by an adversary to induce a certain number of bit flips in the plain text with a high probability $p = P[W_s = W_s^*]$.

Consider now an attacker with a decryption oracle whose goal is producing some particular plain text x^*; this is equivalent to producing some particular residual r^*. The adversary can mount a brute force attack to arrive at that residual by submitting submitting different cipher texts y' such that their syndrome is of the weight determined by $f(W_r)$ in arbitrary order. The probability of success for each of those transmissions is given by:

$$P[r = r^*] = \frac{1}{p * (|x|! - (|x| - W(r^*))!)}$$

For large $W(r^*)$ this formula approaches $\frac{1}{p*|x|!}$; if $|x|$ is also large this approaches $\frac{1}{p*|x|^{W(r^*)}}$, so the complexity of the brute force attack is be $O(|x|!)$ or $O(|x|^{W(r^*)})$; which is polynomial in the message sizel and non-polynomial in the residual weight.

For small $W(r^*)$, the same relation technically holds but this is still insufficient, as the residual weight is not chosen by the designer of the crypto system but by the attacker mounting the attack. In the special case that $W(r^*) = 1$, the attack complexity is **linear in the size of the message**.

3.2.5.1 Mitigations The designer may increase the message size but that provides just polynomial improvements. The designer may also use mitigations like choosing some wide probability distribution or a non-trivial probability distribution; such mitigations reduce the linear factor p in the attack complexity. The designer may even forbid short residual weights ($W_r > W_{rmin} \vee W_r = 0$) in general

using some sort of all-or-nothing construction which would drastically decrease the usefulness of the construction. Even in this case, an adversary could simply move the goal post by allowing extra noise in their target residual if the real-world attack they are trying to mount allows this.

Techniques such as layering extra redundancy on top (explicitly or implicitly) are likely of very limited effectiveness because they also just move the goal post; the complete system will still be susceptible to this attack; the adversary now just needs to increase their syndrome weight to achieve the same residual weight.

3.2.5.2 Consequences Short-Distance Brute-force imposes fundamental limitations on the security of `PMR` in CCA2 attack scenarios. CCA1 scenarios are not affected, so applications that can ensure nonces are "crossed off" and never decrypted are not affected (e.g. real time streaming with a single recipient).

There are some arguments for why this limitation may be acceptable; `CCA2` does not protect against size based attacks which has real world impact [citation needed]. Applications that can deal with lossy channels in the first place might not be affected by the adversaries ability to induce bit flips, but arriving at a formal proof of security for such applications should be exceedingly hard. Finally the mitigations might not get as to NP complexity but can still drastically improve the real world security of any system with `PMR`; ultimately this just means that a lot of extra caution is needed when using `PMR` in such settings.

4 Constructions

By convention in this paper a python-like pseudocode is used to specify algorithms. A parameter `&<name>` denotes a reference to a value. This specification style is intended to be replaced with a formal specification in some established grammar. `clone(v)` where v is a reference denotes taking a copy of the value referenced to; assignment to references assigns to the value.

4.1 Choice of FEC

`fec(\dot)` and `unfec(\dot)` are two polynomial time, deterministic and stateless algorithms such that `unfec` is the inverse of `fec`. The code is assumed to be better at correcting sparse syndromes than dense ones.

An FEC for this particular use case has not been chosen. The precise properties required from the code are at the current time not well understood. In the DDE-CCA2 secure constructions, the code is used to induce diffusion, formalization of the requirements will be guided by that application. Some properties the code should definitely have: Constant time decoder and encoder, space-efficiency, fast decoder and encoder, optimized for randomly distributed errors. Space efficiency is particularly important in `PMR` because the primary motivation behind using `PMR` is space efficiency.

The proper proof of security for the instantiations will largely depend on the specifics of the FEC scheme.

4.2 Choice of RNG

The `rng` parameter in the specifications below denotes some unique random oracle. `clone(rng)` duplicates the state of a random oracle such that $a \leftarrow f(clone(rng)); b \leftarrow f(rng); a = b$ for any function f that is not probabilistic or stateful.

All constructions require instantiation of the random oracle with a random number generator. Which RNG is chosen, is not explicitly mandated by this paper. However, I strongly suggest using either chacha20 or shake256 with appropriate key derivation where applicable.

4.3 The Grind Shuffle

In the specifications below `shuffle(rng, data)` and `unshuffle(rng, data)` denote two stateless, deterministic, polynomial time algorithms such that `unshuffle` is the inverse of `shuffle` provided that the state in both random number generators is the same.

An efficient, cryptographic, bitwise shuffle is required to instantiate the scheme. Standard shuffling methods such as the fisher-yates [citation needed] shuffle are not suitable as these algorithms use key dependent memory accesses, thereby introducing a timing side channel. Cryptographic shuffles such as the thorp-shuffle [citation needed] or the swap-or-not shuffle [citation needed] are slow to [is this actually the case?] implement.

The grind shuffle is a cryptographic shuffle mixing two blocks of data using only rotate and standard bitwise operations. It needs to be seeded with a random number generator (random oracle). The grind shuffle is conjectured to asymptotically approach a truly uniform permutation on location with an increasing number of rounds.

4.3.1 Definition

The grind shuffle takes a random number generator and two blocks of data to be mixed as input. The bits of both blocks are swapped with a random mask, then one or both blocks are rotated by a randomly chosen distance. These two steps are repeated until both blocks sufficiently mixed.

```
def grind_round(&block0, &block1, &rng):
  swap_with_mask(block0, block1, rng.getBytes(blockSize))
  block0 <<<= rng.getInt(0, blockSize) # Random integer in the interval [0; blockSize)
  # block1 <<<= rng.getInt(0, blockSize)
```

Bitwise swap with mask can easily be defined as a bitwise operation (a more efficient definition using XOR can likely be found):

```
def swap_with_mask(&block1, &block2, mask):
  t1 = clone(block1), t2 = clone(block2)
  block1 = (t1 & ~mask) | (t2 & mask)
  block2 = (t2 & ~mask) | (t2 & mask)
```

Each of the operations is fully reversible. Creating the inverse of the shuffle only requires extracting the parameters to `grind_round` from the key stream backwards and applying rotation before swapping.

4.3.2 Constant time rotation

Constant time bitwise rotation within the architecture's word size is a standard operation in most cryptographic software libraries. Its efficient implementation on real hardware is out of scope for this paper. In some situations, constant time rotation by a variable distance may not be supported; possibly because the rotation distance is large. In these cases, it can be implemented as the sum of a number of fixed distance rotations where the distances are powers of two. Rotations that are not needed are masked. Rotating the elements of an array by a fixed amount without a timing side channel can be accomplished with standard techniques.

4.3.3 Correctness argument

During each round, exactly half of the bits are randomly selected for a rotation by a random distance. After r rounds, the rotation distance for a particular bit starting at position i_0 is $i_0 + \sum_{n=0}^{r} d_n * b_n$. d_n. This distance is the same for each bit during each round, but the value of d_n (the switch enabling the rotation for a particular round) is correlated with just the complementary bit. The total space of possible rotation distances after r rounds is $R = 2^r$ so shuffling should be achievable in $O(log(B))$ rounds where B is the block size.

4.3.4 Grind as a structured method of constructing regular permutations

Any permutation can be encoded in the grind framework in a small number of rounds by choosing the appropriate key. This may be useful in the implementation of bitwise permutations. Computing the appropriate key in the general case is an open problem; optimal would be a some function converting the fisher-yates representation of the permutation to the grind representation.

The following are constructions with special properties build on the previously defined grind round function.

4.3.4.1 Shuffling longer vectors While long-distance rotation can be implemented using the techniques outlined in above, it may be more efficient to just split the data into multiple chunks and mix them incrementally. Used like this, the round function mixes the bits from two blocks at a time; like in the two block variant, rotation serves to decorrelate the relative shift of bits from the same block of origin.
In this mode of operation, **both blocks** must be rotated by a random amount, not just the first. Finding an optimal, deterministic mixing pattern remains an open question for now.

4.3.4.2 Shuffling with a mask It is possible to ensure that some bits remain fixed while the rest of the bits are shuffled randomly. In the mask, all bits that should remain in place are set to zero, all others are set to one.

```
def grind_round(&block0, &block1, &rot0, &rot1, &mask0, &mask1, rng):
  swap_with_mask(block0, block1, mask0 & mask1 & rng.getBytes(blockSize))

  dist0 = rng.getInt(0, blockSize)
  rot0 += dist0
  block0 <<<= dist0
  mask0 <<<= dist0

  dist1 = rng.getInt(0, blockSize)
  rot1 += dist1
  block1 <<<= dist1
  mask1 <<<= dist1

def grind(&block0, &block1, mask0, mask1, &rng):
  rot0 = 0
  rot1 = 0
  for _ in range(roundCount):
    grind_round(block0, block1, rot0, rot1, mask0, mask1, rng)
  block0 >>>= rot0 % blockSize
  # mask0 >>>= rot0 % blockSize # Never used again
```

```
  # block1 >>>= rot1 % blockSize
  # mask1 >>>= rot1 % blockSize
```

This works for two reasons: The masked bits are never moved to the other block. They are rotated, but their offset is kept track of through the `rot0`, `rot1` variables. `mask0`, `mask1` is rotated as well, so we can keep track of all the masked bits. After all rounds of shuffling have been reset, their absolute rotation is reset to zero with a right shift.

Note that masking clearly increases the required number of rounds, especially in extreme scenarios like masking most bits in one block but not the other. Masking all bits disables the shuffling altogether. Further analysis is required to find a variant that is not susceptible to these issues.

4.3.4.3 Shuffling bit vectors whose size is not a multiple of the block size

Use padding to achieve the required block size. Use shuffling with mask to shuffle without moving the padding. Discard the padding.

4.4 `dde-cca1-cipher`: A DDE-CCA1-secure construction

Encryption in the random oracle model provides a sound basis to build upon. The FEC is sandwiched in shuffle and encryption operations to mask the location of bits. All operations are linear and use entropy from a random oracle, so a security proof should be achievable.

The cipher design outlined in this section aims to achieve simplicity and provable security. Specifically, the cipher should be implemented as a reduction to a common PRF (e.g., blake2, chacha, keccak) instead of a from scratch construction to achieve provable. Security in the following games is conjectured `IND-CCA1`, `pl-ND-CCA1`, `pl-IND-CCA1`, `FEC-CCA1` and `LEU-CCA1`. For this security level to be realistic, the implementation must reliably prevent the reuse of nonces; this is possible in some streaming scenarios.

`IND-CPA` can be achieved by using unauthenticated encryption in the random oracle model [citation needed]. This scheme provides 1:1 malleability. We could perform FEC on ciphertext or plaintext. This scheme also gives the adversary the ability to induce arbitrary error patterns in the plaintext. This is an extremely efficient construction in terms of both ciphertext size and performance, but is not secure. Still, it provides a good basis to build the full scheme on.

```
def encrypt(&rng, plaintext):
  return plaintext ^ rng.getBytes(len(plaintext))

def decrypt(&rng, ciphertext):
  return encrypt(rng, ciphertext)
```

To achieve the security claims given above, the forward error correction step and the location of bits need to be masked. For this purpose, the forward error correction sandwiched in shuffle operations and forward error correction is used. You may notice that this function does not return an error estimate. It is possible that the error estimate generated by `unfec()` can be directly used.

```
def encrypt(rng, plaintext):
  x = plaintext
  x = x ^ rng.getBytes(len(x))
  x = shuffle(rng, x) # Some cryptographic shuffle
  x = fec(x)
  x = shuffle(rng, x)
  x = x ^ rng.getBytes(len(x))
  return x
```

```
def decrypt(rng, ciphertext):
  rng.seek(bitsNeeded) # Need to consume random bytes in reverse
  x = plaintext
  x = x ^ rng.reverse().getBytes(len(x))
  x = unshuffle(rng, x) # Inverse of shuffle
  (x, w) = unfec(x) # Inverse of FEC
  x = unshuffle(rng, x)
  x = x ^ rng.getBytes(len(x))
  if w > w_max:
    return None
  else
    return (x, w)
```

w_{max} needs to be chosen so there is a large number of invalid messages per valid message.

4.4.1 Security argument

Encryption in the random oracle model is secure despite just using a linear operation on the plaintext: XOR may be linear, but it is used with a large amount of random information. The cipher outlined above should be secure for the same reasons.

The adversary has minimal information about the input and the output of the FEC. Despite knowing the plaintext and ciphertext, they know neither the input values nor the output of the FEC, since those are obscured by the two XOR operations with the key stream. Nor do they know which bits in the ciphertext relate to which bits in the FEC output, or which bits in the plaintext relate to which bits in the FEC input. Since the chosen FEC is better at correcting spares syndromes, the adversary doesn't know the number of bit flips in the plaintext after decryption, since the shuffle obscures the error pattern. The adversary knows just the statistical distribution of errors in the plaintext.

Under the random oracle model, the key stream for one nonce/key combination is independent of any other key stream. Nonce reuse is prohibited either by use of a stateful encryption oracle or by choosing nonces at random from a large space. The recipient needs to make sure no nonce is used for decryption more than once. Assuming an adversary can use a known-plaintext attack to fully recover the key stream used for one encryption, no information about the key stream used for other messages is gained.

An adversary may try to submit cipher texts chosen by the adversary without starting with a ciphertext generated by the encryption oracle; the recipient refuses such probing attempts since the scheme is `LEU-CCA1`-secure. Since w_{max} is chosen such that there are many invalid ciphertexts, the probability of gaining any information using this method is negligible.

4.5 `dde-cca2-cipher-nopunct`: A DDE-CCA2-secure construction

Achieving CCA2 security necessitates the use of techniques from permutation-based encryption, as recovering even some bits from the key stream may enable an attack. The construction is similar to a substitution-permutation network. It uses shuffles and FEC instead of S- and P-boxes. The scheme is conjectured to provide `IND-CCA1`, `pl-IND-CCA2`, `pl-ND-CCA2`, `LEU-CCA2` and `FEC-CCA2` security.

The scheme from the previous section is insecure under adaptive chosen ciphertext attacks; just determining part of the inner and outer shuffle would suffice to enhance the adversary's ability to produce specific error patterns. This structure can be probed using a differential attack. To achieve CCA2 security, it is thus necessary to mask the key stream even under differential attacks.

In non-malleable schemes (i.e., authenticated encryption) CCA2 attacks are reduced to CPA attacks because the authentication tag constitutes a proof that the adversary knows the plaintext. DDE schemes specifically provide ciphertext and plaintext malleability as a feature, so using the authentication technique is out of the question.

Instead of using authentication, techniques from block ciphers are used to obscure the key stream. Shuffle constitutes a fully randomized permutation on location and is used as a P-box. FEC is used to provide diffusion. One round of the cipher is FEC sandwiched in two shuffles. Shuffles from adjacent rounds are merged, since shuffle is a linear operation (two shuffles by a random key just yield a third shuffle by a random key with the *same amount of randomness*). Before and after the rounds of FEC/shuffle, the data is XORed with the key stream.

The error estimate of the last FEC operation is used during decryption.

```
def encrypt(rng, plaintext):
  x = plaintext
  x = x ^ rng.getBytes(len(x))
  for _ in roundCount:
    x = shuffle(rng, x) # Some cryptographic shuffle
    x = fec(x)
  x = shuffle(rng, x)
  x = x ^ rng.getBytes(len(x))
  return x

def decrypt(rng, ciphertext):
  rng.seek(bitsNeeded) # Need to consume random bytes in reverse
  x = plaintext
  x = x ^ rng.reverse().getBytes(len(x))
  for _ in roundCount:
    x = unshuffle(rng, x) # Inverse of shuffle
    (x, w) = unfec(x) # Inverse of FEC
  x = unshuffle(rng, x)
  x = x ^ rng.getBytes(len(x))
  if w > w_max:
    return None
  else
    return (x, w)
```

4.5.1 Security argument

`dde-cca2-cipher-nopin` is an extension of `dde-cca1-cipher`; therefore the same arguments for `DDE-CCA1` security apply here. To achieve DDE-CCA2 security, this scheme provides additional security against differential cryptanalytic attacks.

The proper security proof of the scheme will depend on the properties of the specific FEC. Since the FEC has not been chosen, a model of its regular properties that this scheme is supposed to obscure is needed. The following properties are chosen:

Locality: The location of errors in the code predict the location of errors after decoding, regardless of the actual value of code points or data. This relationship is assumed to be trivial: Errors in start/middle/end of are likely to yield errors at start/middle/end of the data. This choice is justified because locality of data usually leads to increased performance in most CPU architectures so it is likely real world schemes will provide it. It is also a natural choice because it is easy to think about.

Preference for sparse syndromes: The code is assumed to be better at correcting randomly distributed errors than correcting burst errors. While there are many FECs that perform better with bursts, this choice makes sense as the FEC is paired with a shuffle which will spread apart bursts of errors. This assumption also makes sense with locality as to correct burst errors, more distant bits need to be involved in the decoding process than just the close neighbors.

Under a differential attack with low weight syndromes, all bit flips are likely going to be corrected. The adversary keeps adding bit flips, which due to the random shuffle surmounts to flipping random bits in the code. At some point in this process, the syndrome is sufficiently dense that the outer round of FEC is no longer able to correct all errors.

In the case of a single round variant, errors are starting to appear in the plaintext. Due to the outer shuffle, these errors appear at random locations even if the syndrome density is high enough to produce multiple errors close to each other. As the adversary keeps adding errors, some bits in the plaintext will be flipped a *second time*; i.e., they are reset to their original value. While the overall error density in the data keeps increasing, this is not necessarily the case when just considering a few bits in the output. Picture how the error vector changes during decoding: The errors in the data after decoding may jitter around in place slightly, while the average error rate within a window of a few bits keeps increasing. After shuffling the decoded bits, this jitter effect will no longer be localized. Instead, the relative error density may change quite drastically; in some cases, with errors moving location across the entire width of the ciphers block size.

Let us now consider the case of a second layer. In general, the same process as in the first round now repeats with the following exception: While in the input to the first FEC the error density for each window of adjacent bits increases monotonically, the error density in the input to the second round occasionally decreases. Recall, that the FEC chosen is better at correcting errors if the syndrome is sparse, so the changes in error density produced by the first round may actually yield a syndrome that is easier to decode for the second. This effect can outweigh the mere increase in bit flips.

With two rounds or more, adding another error may actually decrease the number of errors after decoding, even if the FEC used doesn't provide such a behavior. Every added round amplifies that effect. Hopefully, an upper bound for the number of rounds required can be derived at by using the fact that the shuffles are fully randomized with a random oracle.

4.6 `dde-cca2-cipher`: A better DDE-CCA2-secure construction

Note that in `dde-cca2-cipher-nopunct`, security, and redundancy are proportional. This is undesirable as this likely will lead to excessive redundancy levels just to achieve a sufficient number of rounds for security. Puncturing (removing some bits from the FEC output) can be used to reduce the redundancy level. Instead of filling those with a constant value during decoding, a second oracle dedicated to masking the decoding process is generated from the ciphertext: $decRNG \leftarrow RNG(k, n, y')$. This construction is similar to a MAC, except that it produces arbitrarily many bits; these bits are also kept private.

This construction provides extra methods of adjusting the cipher parameters:

- Adding more rounds increases security and redundancy
- Puncturing more bits increases security and decreases redundancy
- Using a code with higher redundancy increases redundancy and might increase security

4.6.1 Security argument

The puncturing technique serves to improve security further. Note that the decoding RNG produces arbitrarily many secret bits because it is initialized with the symmetric key and nonce, and its values

are never exposed by the decoder. Note also, that the stream cannot be probed during a CCA2 attack because the derived ciphertext is also used to initialize the RNG.

This provides a vastly more direct way of randomizing the decoding process than relying on the diffusion properties of the cipher. This should drastically reduce the number of rounds required and substantially increase the chances of being able to arrive at a formally proven upper bound for the number of rounds needed.

4.7 `aesaf-cipher`: Authenticated encryption with security against fuzzing

To achieve full `IND-CCA2` security and `FEC-CCA2` on top, `dde-cca2-cipher` is combined with any authenticated encryption scheme.

The inner XOR with the key stream is replaced with the authenticated encryption scheme; in other words: crypto first and FEC on top. The error estimate is ignored; using it to abort early would introduce a timing side channel.

For increased efficiency, a final round of our normal FEC followed by a hard-decision algebraic code should be used to mop up any remaining errors, especially if the `fec()` is suceptible to error floors.

The scheme either returns the valid plaintext or yields an authentication error; this means probing this should be harder than in the partial message scenario. The number of rounds may be adjusted appropriately.

4.7.1 Security argument

The scheme is CCA2 secure since it operates only on the ciphertext generated by a CCA2 secure authenticated encryption scheme. The argument for `FEC-CCA2` security is that this scheme encompasses a `FEC-CCA2` secure scheme.