Accordion mode based on Hash-Encrypt-Hash*

Hieu Nguyen Duy[2], Pablo García Fernández[2], Aleksei Udovenko[2], Alex Biryukov[1,2]

[1] FSTM, University of Luxembourg
[2] SnT, University of Luxembourg
first-name.last-name@uni.lu

Abstract. NIST is planning a call for a tweakable variable-input-length strong pseudorandom permutation (VIL-SPRP), coined as "an Accordion mode". In this paper, we survey tweakable VIL constructions, specially focusing on those based on the Hash-Encrypt-Hash paradigm (including HCTR and similar designs) in terms of their applicability as an Accordion mode, and discuss their efficiency and shortcomings. Furthermore, we also introduce an improved version that achieves Beyond Birthday Bound (BBB) security.

Keywords: Hash-Encrypt-Hash · Accordion Mode · HCTR · ACCOR

Contents

*This research was funded by the Luxembourg National Research Fund (FNR), project CryptoFin C22/IS/17415825. For the purpose of open access, and in fulfilment of the obligations arising from the grant agreement, the authors have applied a Creative Commons Attribution 4.0 International (CC BY 4.0) license to any Author Accepted Manuscript version arising from this submission.

1 Introduction

Awareness about the need for a new mode for block ciphers has risen recently. It would be crucial for some applications to have a mode that is more flexible and robust than existing modes such as AES-GCM. Concretely, NIST is calling for the development of a tweakable variable-input-length strong pseudorandom permutation (VIL-SPRP), coined as "an Accordion mode".

In this work, we survey related constructions, focusing on those based on the Hash-Encrypt-Hash paradigm, and present three instantiations of an accordion mode following it. We evaluate their flexibility and robustness according to the NIST's criteria.

Previous work We briefly mention previous constructions relevant to our work. HCTR is a construction proposed in 2005 [WFW05], and is used as the main basis of inspiration for our construction. It is closely related to other constructions such as HCH [CS06], HSE [MM07], OCB1 ([Rog04]), and XCB [MF04]. Based on these ideas, some generalizations were presented in 2011, remarkably LBC1 (see [MI11]).

Other variants of HCTR have already been developed. In 2018, a tweakable version that achieves beyond birthday bound (BBB) security was proposed, THCTR ([DN18]). Furthermore, another efficient construction was created that year, Adiantum, based on HBSH ([CB18]). Also, a similar 3-round construction achieving BBB security which is based on tweakable block-cipher calls was published that same year, ZCZ (see [BLN18]).

Additionally, other interesting proposals for tweakable block ciphers have been developed. From the original version of tweakable encryption, LRW, several improvements have appeared in the recent years, although they do not have the required properties to be viable candidates for the accordion mode. Other schemes based on deck constructions were proposed ([GDM19]), as well as another one using masking functions, TIE-plus ([Zha23]).

Tables 1 and 2 summarizes all of these constructions (this is a working draft, we welcome feedback and any corrections or typos).

In section 2.2, we have presented another survey, in this case relating those tweakable block ciphers that could be used as primitives for the accordion constructions.

Our contribution After presenting a survey of the most relevant projects in the literature, we focus our attention into three different instantiations of a common scheme. The first one is an ideal construction, used to study theoretical security bounds; the second one is a version of high efficiency, but limited security; the last one, although slightly less efficient, increases greatly the security bound of the second construction.

Outline In Section 2, we explain the HEH paradigm, give a brief introduction to HCTR, and survey the other main results in the area of tweakable block ciphers. The following three sections present our main contributions. Section 3 is about ACCOR-HEH, a general view of the scheme. Section 4 is devoted to ACCOR-S, a concrete instantiation of the general blueprint that achieves great efficiency. The following section, number 5, explains ACCOR-L, a more powerful realization that achieves BBB security, although with a slightly worse perfomance. Afterwards, in Section 6, we show the derived functions and applications associated with this scheme, as well as alternative methods of using it. Then, in Section 7, we comment on the specific security requirements expected of an accordion mode.

In the appendices we give a detailed proof of the security bound of the ideal construction, as well as additional suggestions to improve the flexibility, and details about security characteristics of the other schemes.

Table 1: Summary of the main VIL (tweakable) block ciphers in the literature

Name	Year	Security	Type of hash	Efficiency	Other characteristics
EME [HR04]	2003	$7q^2a^2/2^n$		$(2m+2)$BC	
CMC [HR03]	2003	$7q^2a^2/2^n$		$(2m+1)$BC	
XCB [MF04]	2004	$8q^2a^2/2^n$	Polynomial	$(m+1)\text{BC}+2\text{HF}$	
OCB1 [Rog04]	2004	$9.5q^2a^2/2^n + 2^{n-\tau}/2^n$		$(m+2)$BC	
HCTR [WFW05]	2005	$q^3a^2/2^n$	Polynomial	$m\text{BC}+2\text{HF}$	
HCH [CS06]	2006	$7q^2a^2/2^n$	Polynomial	$(m+3)\text{BC}+2\text{HF}$	
HSE [MM07]	2007	$5q^2a^2/2^n$	ϵ-AXU	$m\text{BC}+2\text{HF}$	
LBC1 [MI11]	2011	$3q^2a^2/2^{2n}$	Polynomial, ϵ-AXU, $\epsilon \approx 2^{-2n}$	$(m-1)(n,n) - \text{TBC}+4\text{HF}$	
TCT$_2$ [ST13]	2013	$qa^2/2^n + 6a^3q^3/(2^{2n-2} - a^3q^3) + 1296q^3/(2^{2n-2} - 216q^3)$	Polynomial, ϵ-AU/ $\epsilon - \text{AXU}_2$	$(6m+6)\text{BC}+ (7\text{m}+7)\text{HF}$	Two different hash functions
ZMAC [IMPS17]	2017	$2.5q^2a^2/2^{n+min\{s,n\}} + 4q/2^n$		$(6+m)$TBC	There is an unknown constant in the security bound
Adiantum [CB18]	2018	$q^2a/2^{116}$	Polynomial, ϵ-ΔU	BC + SC (output: $m-1$ blocks)+2HF	SC: XChaCha12
ZCZ [BLN18]	2018	$3q^2a^2/2^{2n+1}$	Polynomial	$(2m+6+m/n)\text{TBC}+4\text{HF}$	Mix tweak and key
THCTR [DN18]	2018	$2qa/2^n + 2qa^2/2^{n+s^*}$	n-bit keyed AXU + $(n+s^*)$-bit keyed pAXU	$m\text{TBC}+3\text{HF}$	Hashed tweak. No reused tweak
Deck-based [GDM19]	2019	$q^2/2^{127}$	ϵ-XU	$2\text{VIL}-\text{SC}+2\text{HF}$	Better security if no reused tweak
HCTR2	2021	$1.5q^2a^2/2^n$	Polynomial, AXU	$m\text{BC}+2\text{HF}$	
TIE-plus [Zha23]	2022	$q^2a^2/2^n\ell + q^2a^2(1-1/\ell)/2^n + 2q^2a/2^n$	(ϵ,μ,δ)-masking function	$2m\text{MF}+\text{mP}$	Provable multi-key security

HF: Hash Function, BC: Block Cipher, TBC: Tweakable Block Cipher, SC: Stream Cipher, VIL-SC: Variable-Input-Length Stream Cipher, P: Permutation, MF: Masking Function, AXU: Almost XOR Universal, AU: Almost Universal, ΔU: Almost Δ Universal, pAXU: partial AXU, XU: XOR Universal, n: block size, a: maximum message length in blocks, q: number of queries, s: tweak length, s^*: tweak length after hashing, m: number of blocks of a specific message, ℓ: number of keys, τ: tag length

2 The Hash-Encrypt-Hash paradigm

One of the methods used in the past to design tweakable block ciphers consist on this three-layered scheme known as HEH.

In this scheme, first there would be a hash of some or all of the input message and the tweak. This hashed value would be later used in the encryption of the message, via a block cipher, a stream cipher, or some combination.

This idea, while very basic, allows for great flexibility. In future sections we will show how it can be used to construct a secure tweakable block cipher without losing efficiency.

2.1 HCTR

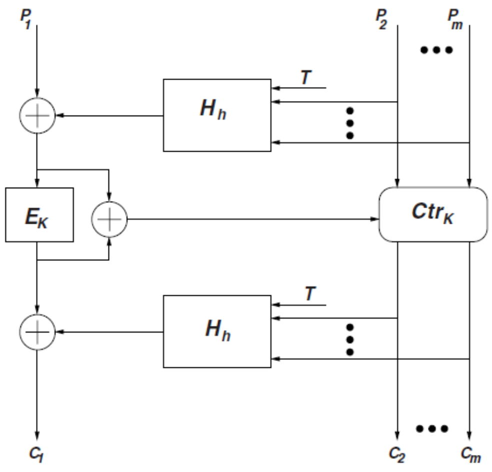


Figure 1: Scheme of the HCTR construction (This image comes from [CN08])

This construction, one of the most basic constructions following the HEH paradigm, is based on the next scheme ([WFW05]):

1. Divide the plaintext in one first block M, and the rest of the blocks N
2. Hash N together with the tweak
3. XOR the result with M to obtain M'
4. Encrypt M', and XOR the result with it to obtain S
5. Use S as the counter for a counter mode
6. XOR the resulting stream with N to obtain all the ciphertext blocks, except for the first one
7. Use again the same hash function with the tweak and the last part of the ciphertext, and XOR the result with the encryption of M' to obtain the first block of the ciphertext

The hash function proposed is polynomial, AXU, and considers the length of the input as one of the coefficients. As the underlying block cipher, we could use AES_{256} or AES_{128}. Furthermore, the tweak length is fixed.

The security of this construction is below the BBB, and it is affected greatly by the maximum message length allowed. It is important to remark that, at least partially, this is due to the fact that E and the counter mode use the same key.

2.2 Survey of results

Table 2: Summary of the main tweakable block ciphers in the literature

Name	Year	Security	Efficiency
		Custom construction	
Threefish [FLS+09]	2009	256 bits	72P + 288MIX
PRINCE [BCG+12]	2012	$q/2^{126}$	12SB + 11LL
TWEAKEY [JNP14]	2014	64 bits (at least)	rP + rTSUP
SKINNY [BJK+16]	2016	88.5 bits	40 rounds AES-like
QARMA [Ava17]	2017	176 bits	46P + 24SB + 23MC + 22LFSR
PRINCEv2 [BEK+20]	2020	112 bits	12SB + 11LL
		Block Cipher based	
LRW0 [LRW02]	2002	$q^2/2^n$	2BC
LRW1 [LRW02]	2002	$3q^2/2^n$	1BC + 2HF
SBC [MI11]	2011	$6.5q^2/2^{n+s}$	$2(n,s) -$ TBC + 2HF
XHX [JLM+17]	2017	$3(s+2)q^2/2^{n+k} + 4q/2^k$	1BC + 3HF
XHX2 [LL18]	2018	$1364q^{3/2}/2^{2n} + 4096q^3/2^{4n}$	2BC + 4HF
LRW2 [JN20]	2020	$3q^2/2^n$	1BC + 2HF
CLRW2 [JN20]	2020	$54q^4/2^{3n} + 2q^2/2^{3n/2}$	2BC + 4HF
TNT [JKNS24]	2023	$q^2/2^n$	3BC
LRW+ [JKNS24]	2023	$2q^2/2^{3n/2} + 32q^4/2^{3n}$	2BC + 2HF

P: Permutation, SB: S-Box, MC: MixColumns-like function, MIX: non-linear Mixing function, TSUP: Tweakey State Update function, LL: Linear Layer, LFSR: Linear-Feedback Shift Register, BC: Block Cipher, TBC: Tweakable Block Cipher, HF: Hash Function, q: number of queries, n: input length of the underlying block cipher, s: tweak length, r: number of rounds, k: key length

We provide a summary of similar HEH constructions in Table 1. As we can see, there are several constructions that achieve BBB security, although they lack in efficiency. From these constructions, the most efficient seems to be ZCZ [BLN18], which employs adequately chosen primitives to obtain the best balance between security and performance.

Regarding those that do not achieve BBB security, LBC1 [MI11] seems interesting, as well as Adiantum [CB18]. The former employs a polynomial hash function and maintains good efficiency, while the latter presents an interesting scheme that combines three fundamental components of symmetric cryptography: hash functions, stream ciphers, and block ciphers.

Additionally, we have surveyed some tweakable block ciphers proposed in the literature that could work as primitives inside the variable-input-length constructions. These are summarized in table 2. The first part of the table includes those that developed their own encryption system, while the other ones use an underlying block cipher, like AES. We are more interested in the latter. The best security comes from those based in cascading LRW.

3 ACCOR-HEH

In this section, we are going to present the general idea of our scheme, the ideal theoretical construction, presented in Fig. 2. In following sections, we will present two specific instantiations, one of them achieving BBB security.

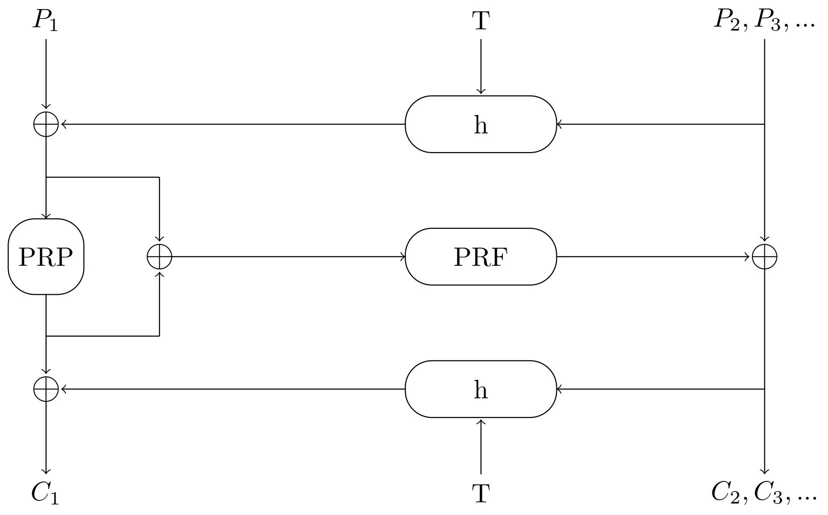


Figure 2: Scheme of ACCOR-HEH

Encryption algorithm:

1. Hash together the tweak and all the plaintext, except for the first block
2. XOR the resulting hashed value with the first plaintext block to obtain P'
3. Pass P' through a PRP (later instantiated by a block cipher) to calculate a new value, C'
4. XOR P' and C' to find S
5. Input S (possible truncated to an appropriate size) into a PRF (later instantiated by a stream cipher) to obtain a stream, which is XORED with all the plaintext, except for the first block, to find the corresponding, second part of the ciphertext
6. Hash together the tweak with all of this ciphertext, and XOR the resulting hashed value with C' to obtain the first ciphertext block

This scheme is symmetric. Therefore, the decryption function is equivalent to this algorithm. It is important to remark that the corresponding key associated with the block cipher must be different to the one used in the stream cipher.

In case the input is not a multiple of the input size of the underlying hash function, we would create an internal padding just for the hash. However, the ciphertext would maintain the original size of the plaintext.

With respect to the security of this construction, we have the following theorem:

Theorem 1. *Considering an adversary that performs q queries, with a total (padded) message length in blocks σ, and in a time at most t,*

$$\begin{aligned} \boldsymbol{Adv}^{\mathit{TVPERM}}_{\mathit{ACCOR\text{-}HEH}}(q,\sigma,t) \leq \frac{q^2}{2}(\frac{2}{2^n}+\frac{1}{2^{n_I}}) + \sum_{1\leq i<j\leq q} \epsilon_{i,j} \\ + \mathit{PRF}\boldsymbol{Adv}^{\mathit{FUNC}_{n_I,n_O}}_{\mathit{SC}}(q,\sigma-q,t+O(\sigma)) \\ + \mathit{SPRP}\boldsymbol{Adv}^{\mathit{PERM}_n}_{\mathit{BC}}(q,t+O(\sigma)) \end{aligned} \tag{1}$$

where n is the size of the block cipher input, n_I is the size of the stream cipher input, n_O the size of the stream cipher output, and $\epsilon_{i,j}$ is an upper bound on the probability of hash collision guaranteed by an almost universal hash family.

More details can be found in the Appendix A.

4 ACCOR-S

In this section, we present the full construction of our main idea, ACCOR-S. We will be working over the finite field $\mathbb{F}_{2^{128}}$ represented by $\mathbb{F}_2[x]/(p(x))$, with

$$p(x) = x^{128} + x^7 + x^2 + x + 1$$

being the irreducible polynomial over $\mathbb{F}_2$ used, in particular, in the AES-GCM (see [Ser98], [MV05]). Most computations with numbers in this work have to be considered over this field. For brevity, while abusing the notation, we represent the elements of $\mathbb{F}_{2^{128}}$ by integers, whose binary representation describes the coefficients of the respective polynomial in $\mathbb{F}_2/\langle p(x)\rangle$. For example, given ω a root of $p(x)$ and, therefore, a generator of $\mathbb{F}^*_{2^{128}}$,

$$5 = (101)_2 = \omega^2 + 1.$$

We denote the block cipher AES encryption (resp. decryption) of a plaintext x with ℓ-bit key K by $\mathrm{AES}^{\mathrm{E}}_{\ell}(\mathrm{K},\mathrm{x})$ (resp. $\mathrm{AES}^{\mathrm{D}}_{\ell}(\mathrm{K},\mathrm{x})$).

The method that we are going to use to initiate the appropriate sub-keys is the following:

Key schedule

INPUT: 256-bit master key K_I
OUTPUT: 128-bit key K_x, 256-bit key K_y, 256-bit key K_z

1. Compute
$$K_x = \mathrm{AES}^{\mathrm{E}}_{256}(\mathrm{K_I}, 1).$$
For security, we require that K_I leading to $K_x = 0$ is not allowed.

2. Compute
$$K_y = \mathrm{AES}^{\mathrm{E}}_{256}(\mathrm{K_I}, 2)||\mathrm{AES}^{\mathrm{E}}_{256}(\mathrm{K_I}, 3).$$

3. Compute
$$K_z = \mathrm{AES}^{\mathrm{E}}_{256}(\mathrm{K_I}, 4)||\mathrm{AES}^{\mathrm{E}}_{256}(\mathrm{K_I}, 5).$$

Before seeing the encryption method, to improve the security against tweak-related attacks, we propose the option to convert the potentially long variable length tweak into a fixed length tweak. One could either encrypt a non-zero constant 256-bit message (for example, derived from the digits of Pi) with the long tweak using the same scheme and to treat the resulting 256-bit ciphertext as a new fixed length tweak, or to use a secure 256-bit output $\mathrm{MAC}(\mathrm{K_I}, \mathrm{T})$.

Also, given such hashed tweak $T = (T_h, T_l)$, $|T_h| = |T_l| = 128$, an alternative way of producing sub-keys from the master key K_I would be as follows:

$$K_x = \mathrm{AES}^{\mathrm{E}}_{256}(\mathrm{K_I}, \mathrm{T_h} + 1) \quad (2)$$
$$K_y = \mathrm{AES}^{\mathrm{E}}_{256}(\mathrm{K_I}, \mathrm{T_h} + 2)||\mathrm{AES}^{\mathrm{E}}_{256}(\mathrm{K_I}, \mathrm{T_l} + 3) \quad (3)$$
$$K_z = \mathrm{AES}^{\mathrm{E}}_{256}(\mathrm{K_I}, \mathrm{T_h} + 4)||\mathrm{AES}^{\mathrm{E}}_{256}(\mathrm{K_I}, \mathrm{T_l} + 5). \quad (4)$$

This way the keys depend on the tweak as well. This might be one of the ways to go towards BBB security, and against context commitment attacks.

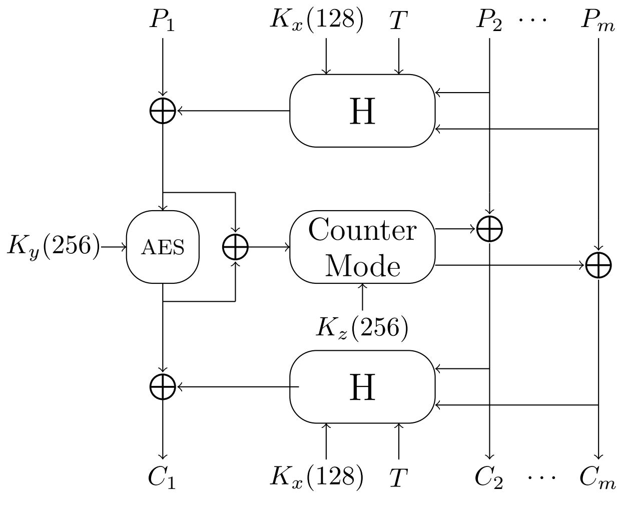


Figure 3: Scheme of ACCOR-S

Encryption function

INPUT: Plaintext P, Tweak T, XUH Key K_x, Block Cipher Key K_y, Stream Cipher Key K_z
OUTPUT: Ciphertext C

1. Divide the plaintext in blocks of size 128 bits. The last block will be (internally) padded. Then, we append the bit-length of the plaintext as an additional block. We require that the tweak has a fixed size. Now, we have the plaintext as
$$P = P_1||P_2||\cdots||P_{m-1}||P_m^*,$$
and the tweak as
$$T = T_1||T_2||\cdots||T_s,$$
where each block has size 128 bits, and $P_m^* = P_m||r$, with r a random string of bits of the appropriate size.

2. Hash the tweak and all except the first block of the plaintext in the following way:
$$P' = P_1 \oplus \left(\bigoplus_{i=1}^{s} T_i K_x^{i+m} \oplus \bigoplus_{i=2}^{m} P_i \cdot K_x^i \oplus \text{bitsize}(P||T) \cdot K_x \right).$$
Here the tweak length is fixed after hashing the variable length tweak, so there is no ambiguity on where the message ends and where the tweak starts.

3. Compute
$$C' = \mathrm{AES}_{256}^{\mathrm{E}}(\mathrm{K_y}, \mathrm{P}').$$

4. Now, let
$$S = P' \oplus C'.$$

5. Use the counter mode with AES_{256}, and the counter given by S, to create $m-1$ blocks of random bits:
$$X_i = \mathrm{AES}_{256}^{\mathrm{E}}(\mathrm{K_z}, \mathrm{S} + \mathrm{i}),$$
for i from 2 to m.

6. As in a one-time pad, XOR the random blocks X_j with the plaintext blocks to find the ciphertext,
$$C_j = X_j \oplus P_j,$$
for j from 2 to $m-1$, and
$$C_m = X_m[0 : len(P_m)] \oplus P_m,$$
where $X_m[0 : len(P_m)]$ consists of the first part of X_m, of length the same as P_m.

7. To compute the first ciphertext block, just calculate
$$C_1 = C' \oplus \left(\bigoplus_{i=1}^{s} T_i K_x^{i+m} \oplus C_m^* \cdot K_x^m \oplus \bigoplus_{i=2}^{m-1} C_i \cdot K_x^i \oplus \text{bitsize}(C||T) \cdot K_x \right),$$
where $C_m^* = C_m||r$, with r of the needed length to form a full block.

Since the scheme is symmetric, the decryption function coincides with the previous algorithm.

4.1 Security

According to the paper [CN08], if we have an adversary $\mathcal{A}$ making q queries of plaintext-ciphertexts blocks, either to the encryption or to the decryption function (the adversary could even query both in a given attack), having at most qa many blocks queried, trying to distinguish this construction from a random permutation, and running in time at most D, then there is another adversary $\mathcal{B}$ over AES_{256}, making q queries to AES_{256} in the same conditions as $\mathcal{A}$, running in time at most $D' = D + O(qa)$, that satisfies

$$\mathrm{Adv}_{\mathrm{HCTR[AES_{256}]}}(\mathcal{A}, \mathrm{a}, \mathrm{D}) \leq \mathrm{Adv}_{\mathrm{AES_{256}}}(\mathcal{B}, \mathrm{a}, \mathrm{D}') + \frac{4.5 \cdot \mathrm{q}^2\mathrm{a}^2}{2^{\mathrm{n}}}.$$

As can be seen in Appendix A, this parameter can be improved by a constant factor.

Specific parameters Assuming an adversary that does q queries to the accordion scheme (possibly with varying tweaks), and limiting each message length to 2^{20} 128-bit blocks, we have that the advantage is, at most,

$$\mathrm{Adv}_{\mathrm{AES_{256}}}(\mathcal{B}, \mathrm{a}, \mathrm{D}') + \frac{4.5 \cdot \mathrm{q}^2 \cdot 2^{40}}{2^{128}} \approx \frac{\mathrm{q}^2}{2^{85.83}} + \mathrm{Adv}_{\mathrm{AES_{256}}}.$$

For a concrete example, if $q = 2^{25}$ queries (i.e. 2^{45} 128-bit blocks) under the same key are made, and $\mathrm{AES}_{256}(k, .)$ is indistinguishable from a random permutation, the attacker will have little chance (bounded by 2^{-35}) to break the construction.

Importance of the specific details of the scheme We present a possible attack that could be performed against similar constructions where the hash function has an ambiguous behaviour, although it is not applicable to our scheme, thanks to our special treatment of the tweak.

Let us suppose that the adversary asks for the encryption of both $P^1 = 0||0$, $T = 0$; and $P^2 = 0||0||0$ (no tweak). Then, since, in both cases, $P' = 384K_x$, we have the same C' and S, which implies that $P_2^1 = X_2 = P_2^2$. Thus, the adversary can distinguish this scheme from a variable-input-length, tweakable family of random permutations.

Furthermore, we also have that

$$\begin{aligned} P_3^2 &= X_3, \\ P_1^1 &= \mathrm{AES}(384\mathrm{K_x}) \oplus \mathrm{X_2 K_x^2} \oplus 384\mathrm{K_x}, \\ P_1^2 &= P_1^1 \oplus X_3 K_x^3. \end{aligned}$$

Therefore,

$$K_x = \sqrt[3]{\frac{P_1^2 \oplus P_1^1}{X_3}},$$

which is easily computable by the adversary.

Another way to avoid this attack consists in using a different hash function that handles the length of both tweak and message in a unique way (for example, considering the coefficient bitsize(P)||bitsize(T)).

4.2 Efficiency

This construction has a similar performance to the GCM mode of AES. In fact, for a given input size, it uses the same number of AES encryptions, as well as two hash function calls in total.

Furthermore, this method is highly parallelizable. After computing the first encryption of the secret parameter P' (which can also be parallelized), one can make all the other calls to the AES function in parallel. Nevertheless, it cannot be computed on the fly (which probably has to be a property of any accordion scheme acting as a large-block block-cipher where every bit of output should depend on all the bits of the input) because it requires seeing all the blocks to compute P'. Thus, it is not optimal for some applications like streaming.

5 ACCOR-L

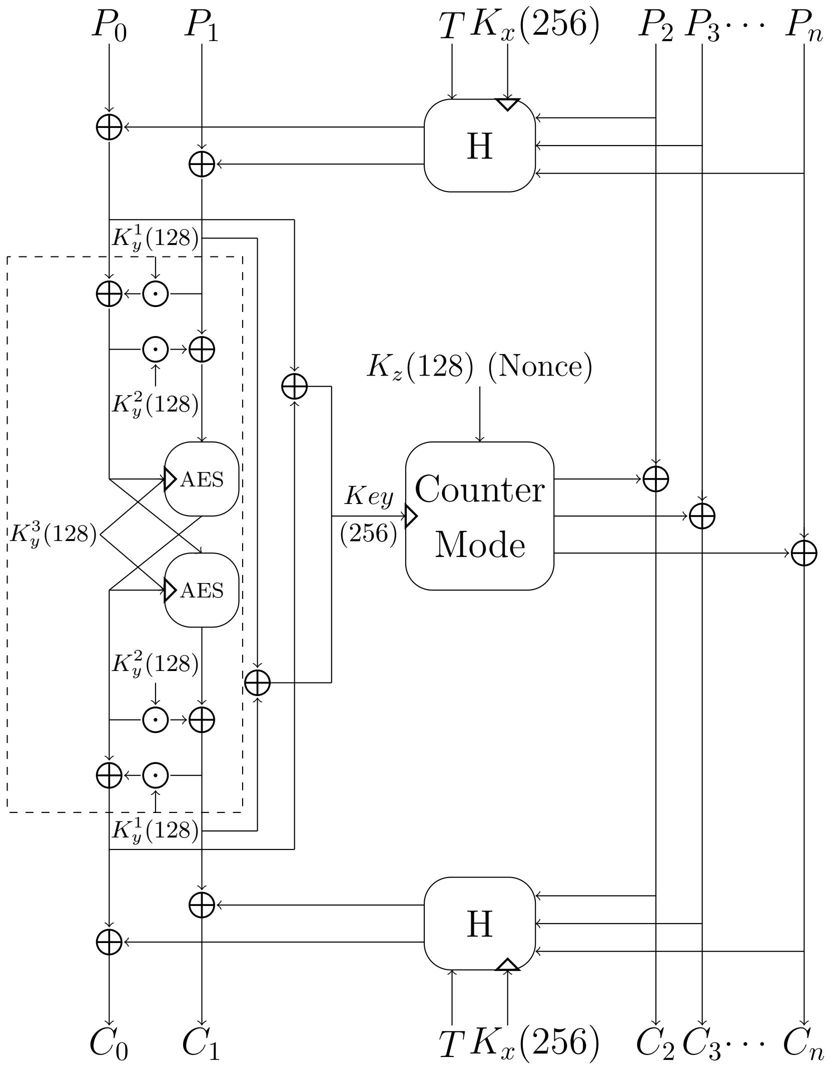


Figure 4: Scheme of ACCOR-L. The dashed rectangle corresponds to the PRP part of ACCOR-HEH.

To increase the security of the scheme, we proposed the construction given in figure 4. Here, the hash function acts over blocks of size $2n$, which forces its output to be divided before later use. Furthermore, we force the key of the counter mode to depend in both the plaintext and the tweak. Therefore, we reduce greatly the probability of unwanted collisions in its output.

The block cipher part of the construction is composed of a first round where a pair of keys are multiplied with each one of the first two plaintext blocks, and the result xored with the other; an AES_{256}-round; and a final round symmetrical to the first one.

The stream cipher is based on a counter mode with AES_{256}, where the key is the concatenated outputs of the block cipher, and the nonce is an extra key. Additionally, each 16 blocks, the key of the counter mode is changed by adding one, which allows to extend greatly the functionality of the cipher, since we increase the security of the system by making the cipher to behave in a more similar way to an SPRP.

As in the previous constructions, the plaintext does not need to have as size a multiple of the input size of the underlying functions. The last block is only padded internally to be used in the hash function. Moreover, the ciphertext has the same size as the original plaintext.

The security of this method is better than that of ACCOR-S. Given the same parameters as in subsection 4.1, the advantage of any adversary should be

$$\mathrm{Adv}_{\mathrm{ACCOR-L}}(\mathcal{A}) \leq \frac{\mathrm{qa}}{2^{124}} \cdot (1 + \frac{\mathrm{q}}{2^{132}}) + \mathrm{Adv}^{\mathrm{RK}}_{\mathrm{AES}_{256}},$$

where a is the maximum message length in blocks, and $\mathrm{Adv}^{\mathrm{RK}}_{\mathrm{AES}_{256}}$ is the related-key security of AES_{256}. Thus, up to $2^{84}/a$ queries (possibly under different tweaks) can be done while keeping success probability of the adversary below 2^{-40}.

If we increment the key of CTR every step, the security bound improves to:

$$\mathrm{Adv}_{\mathrm{ACCOR-L}}(\mathcal{A}) \leq \frac{\mathrm{q}^2\mathrm{a}}{2^{256}} + \mathrm{Adv}^{\mathrm{RK}}_{\mathrm{AES}_{256}}.$$

Such scheme would be less efficient, but closer to SPRP if the underlying cipher was ideal. However simple tweaks to AES master keys need to be carefully checked in light of weak-key schedule and related key attacks on AES.

6 Application modes

6.1 Authenticated Encryption with Associated Data

We can make this construction into an AEAD by treating the tweak as a combination of the nonce and the associated data, and including a last block in the plaintext that ends in the message 10^τ. Due to the security properties of the accordion mode, we know there is no loss of security due to a misuse of the nonce. Furthermore, since the ciphertexts are almost indistinguishable from random, an adversary would need around $2^{\tau+1}$ different ciphertexts until finding one that succeeds in decrypting into a valid plaintext.

In the original HCTR-like scheme this method could not be used because the tweak had to have a fixed size.

6.2 Tweakable Encryption

This method works efficiently for tweakable encryption. Changing the tweak is highly efficient (you only need to do a few new multiplications), unless you decide to use the option of pre-encrypting the tweak, or considering tweakeys. Anyway, changing the tweak for long messages is amortized due to fixed cost of rekeying. Furthermore, there does not seem to be any security flaw with respect to chosen tweak attack due to KDF usage.

6.3 Deterministic Authenticated Encryption

As done in subsection 6.1, we can create DAE by, after padding the plaintext, adding a block ended in the sequence 10^τ, where τ is the authentication security parameter.

This application, which does not require any tweak, is easily introduced in such scheme since eliminating the tweak is not only easy but improves the method's performance slightly.

6.4 Double block size of the underlying block cipher

As in the construction of subsection 5, we can easily accommodate the system to allow an underlying block cipher that accepts 256-bit inputs.

6.5 Improved efficiency

One way to improve the scheme's efficiency is to change the polynomial hash function for a faster, while still secure, version. For example, Poly-1305 ([Ber05]) with AES, or NH ([BHK+99]), could be used instead.

7 Additional security properties

Table 3: Main additional properties of our three schemes

Property	ACCOR-HEH	ACCOR-S	ACCOR-L
TS	UL	UL	UL
Min ML	1 block	1 block	2 blocks
Max ML	UL	2^{20} blocks	2^{30} blocks
MKS	Yes	Yes	Yes
CtxCom	Yes	Yes	Yes
KDIS	Yes	Yes	Yes
NMR	Yes	Yes	Yes
NH	Yes	Yes	Yes
PAS	Yes	Yes	Yes
KCKS	?	?	?

TS: Tweak Size, ML: Message Length, MKS: Multi-key Security, CtxCom: Context Commitment Security, KDIS: Key-Dependent Input Security, NMR: Nonce-misuse resistance, NH: Nonce Hiding, PAS: Padding Attacks' Security, KCKS: Known/Chosen Key Security, UL: Unlimited

7.1 Tweak size

These three construction allow any size for the tweak, preferably multiples of 128 bits.

The AEAD method does not change the requirements on this parameter, even if we include the additional data as part of the tweak. Other option consists in hashing this data into the fixed-length tweak.

7.2 Message lengths

We propose a minimum message length of 128 bits ($a = 1$) for the first two constructions, 256 bits ($a = 2$) for the last one. In ACCOR-HEH, we do not establish a maximum for this parameter, since the security depends only slightly on it. However, in ACCOR-S, this is a key parameter for the security of the system. Therefore, we set a maximum message length given by $b = 2^{20}$ blocks (16.78 megabytes). Other maxima could also be valid, although it is important to realize the close relationship between the message length and the encryption's security.

Due to the relationship of this parameter with the security of ACCOR-L, we establish the maximum for it in $b = 2^{30}$ blocks (137.44 gigabytes).

7.3 Multi-user security

According to **Theorem 1**, and **Lemma 3** of [LMP17], we can prove that the polynomial hash function does not exhibit multi-key security degradation.

We believe this result can be extended to the rest of the scheme, and to the three variants.

7.4 Key and Context Commitment

When using one of the authenticated modes, we have to be careful due to the possibility that there are some ways of finding specific keys or tweaks that allow an adversary to create a valid ciphertext.

Let us consider the following attack: given

$$C_1, \ldots, C_m = \mathrm{HCTR}(\mathrm{T}, \mathrm{K_x}, \mathrm{K_e}, \mathrm{P_1}, \ldots, \mathrm{P_m}),$$

given $Q_1, \ldots, Q_m$ a different plaintext, find T', K_x', K_e' such that

$$\mathrm{HCTR}(\mathrm{T'}, \mathrm{K_x'}, \mathrm{K_e'}, \mathrm{Q_1}, \ldots, \mathrm{Q_m}) = \mathrm{C_1}, \ldots, \mathrm{C_m}.$$

Let us suppose that AES is resistant against these attacks: given $B = \mathrm{AES}(\mathrm{K}, \mathrm{A})$, there is no feasible way to find a different plaintext A', and a key K', such that $B = \mathrm{AES}(\mathrm{K'}, \mathrm{A'})$.

Thanks to this property of AES, to find C_j, for $j > 1$, since

$$C_j = P_j + X_j = P_j + \mathrm{AES_{256}}(\mathrm{K_e}, \mathrm{S} + \mathrm{j}),$$

we need the same K_e and S, which, in turn, fix P_j for $j > 2$.

Now, to find S, since $S = P' + \mathrm{AES_{256}}(\mathrm{K_e}, \mathrm{P'})$, the adversary just have to find appropriate tweak and key that lead to the same P'. Let us see how this can be done.

We have to solve the system of equations:

$$\begin{aligned} f(K_x', T') + g(K_x', Q) &= P' \\ f(K_x', T') + g(K_x', C) &= C', \end{aligned}$$

with both f and g polynomials. Therefore, we just have to find a root for the polynomial

$$p(K_x') = g(K_x', Q) - P' + C' - g(K_x', C).$$

Once this is done, the solution is found by solving the linear equation

$$f(K'_x, T) = P' - g(K'_x, Q),$$

where K'_x, P' and Q are fixed.

To sum up, a forgery can be done, but only affecting the first plaintext block. This forgery is feasible due to the polynomial structure of the hash function. A different type of hash function would probably solve this issue.

Fortunately, this attack is prevented by the improved Key Derivation Protocol explained in section 4.

7.5 Key-Dependent Input Security

Unless the adversary asks to encrypt the decryption of some keys, any other key function should give no information, since the corresponding ciphertext is computationally indistinguishable from random.

There is some sense in which this construction does not achieve this property. For example, if the message to be encrypted is a function of the key such that the corresponding X_j, for some index j, is a clear invertible function of some key, we can recover that key.

7.6 Nonce-misuse resistance

Since the nonce would be inside the tweak, which is considered in the same way as the plaintext, both P' and K_x are secure as long as the adversary cannot recover the plaintext. Therefore, there is no security problem if the same tweak is used more than once, apart from the fact that encrypting twice a message with the same tweak produces the same ciphertext.

7.7 Nonce hiding

Since, in this construction, the nonce is considered part of the tweak, this can easily be hidden (by treating it as plaintext), and the security of this encryption is the same as that of any other plaintext block.

7.8 Padding attacks

In the basic method, there is no wrong ciphertext, which ensures resistance against this type of attack. In the authenticated method could be some problems, none that we know about.

7.9 Known/Chosen Key security

The scheme is easy to distinguish from an ideal cipher if the adversary can know or even choose the keys. This is expected since this is just a 3-round unbalanced Feistel scheme. This can be improved by altering the structure and adding more rounds, but losing efficiency.

References

[Ava17] Roberto Avanzi. The qarma block cipher family. almost mds matrices over rings with zero divisors, nearly symmetric even-mansour constructions with noninvolutory central rounds, and search heuristics for low-latency s-boxes. *IACR Transactions on Symmetric Cryptology*, pages 4–44, 2017. URL: https://tosc.iacr.org/index.php/ToSC/article/view/583, doi:10.13154/tosc.v2017.i1.4-44.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçın. Prince – a low-latency block cipher for pervasive computing applications. *Advances in Cryptology – ASIACRYPT 2012. Lecture Notes in Computer Science*, 7658:208–225, 2012. URL: https://link.springer.com/chapter/10.1007/978-3-642-34961-4_14, doi:10.1007/978-3-642-34961-4_14.

[BEK+20] Dušan Božilov, Maria Eichlseder, Miroslav Kneževic, Baptiste Lambin, Gregor Leander, Thorben Moos, Ventzislav Nikov, Shahram Rasoolzadeh, Yosuke Todo, and Friedrich Wiemer. Princev2 - more security for (almost) no overhead. *Cryptology ePrint Archive, Paper 2020/1269*, 2020. URL: https://eprint.iacr.org/2020/1269.

[Ber05] Daniel J. Bernstein. The poly1305-aes message-authentication code. *Fast software encryption: 12th international workshop. Lecture Notes in Computer Science*, 3557:32–49, 2005. URL: https://cr.yp.to/mac/poly1305-20050329.pdf.

[BHK+99] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. Umac: Fast and secure message authentication. *Advances in Cryptology – CRYPTO'99. Lecture Notes in Computer Science*, 1666:216–233, 1999. URL: https://link.springer.com/chapter/10.1007/3-540-48405-1_14, doi:10.1007/3-540-48405-1_14.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The skinny family of block ciphers and its low-latency variant mantis. *Advances in Cryptology – CRYPTO 2016. Lecture Notes in Computer Science*, 9815:123–153, 2016. URL: https://link.springer.com/chapter/10.1007/978-3-662-53008-5_5, doi:10.1007/978-3-662-53008-5_5.

[BLN18] Ritam Bhaumik, Eik List, and Mridul Nandi. Zcz – achieving n-bit sprp security with a minimal number of tweakable-block-cipher calls. *Advances in Cryptology – ASIACRYPT 2018. Lecture Notes in Computer Science*, 11272:336–366, 2018. URL: https://link.springer.com/chapter/10.1007/978-3-030-03326-2_12, doi:10.1007/978-3-030-03326-2_12.

[CB18] Paul Crowley and Eric Biggers. Adiantum: length-preserving encryption for entry-level processors. *Cryptology ePrint Archive*, 2018. URL: https://eprint.iacr.org/2018/720.

[CDD+24] Yu Long Chen, Michael Davidson, Morris Dworkin, Jinkeon Kang, John Kelsey, Yu Sasaki, and Meltem Sönmez Turan. Proposal of requirements for an accordion mode. *Discussion Draft for the NIST Accordion Mode Workshop 2024*, 2024. URL: https://csrc.nist.gov/files/pubs/other/2024/04/10/proposal-of-requirements-for-an-accordion-mode-dis/iprd/docs/

proposal-of-requirements-for-an-accordion-mode-discussion-draft.pdf.

[CN08] Debrup Chakraborty and Mridul Nandi. An improved security bound for hctr. *Fast Software Encryption. Lecture Notes in Computer Science*, 5086:289–302, 2008. URL: https://link.springer.com/chapter/10.1007/978-3-540-71039-4_18, doi:10.1007/978-3-540-71039-4_18.

[CS06] Debrup Chakraborty and Palash Sarkar. Hch: A new tweakable enciphering scheme using the hash-encrypt-hash approach. *INDOCRYPT 2006. Lecture Notes in Computer Science*, 4329:287–302, 2006. URL: https://link.springer.com/chapter/10.1007/11941378_21, doi:10.1007/11941378_21.

[DN18] Avijit Dutta and Mridul Nandi. Tweakable hctr: A bbb secure tweakable enciphering scheme. *Progress in Cryptology – INDOCRYPT 2018. Lecture Notes in Computer Science*, 11356:47–69, 2018. URL: https://eprint.iacr.org/2019/1324.pdf, doi:10.1007/978-3-030-05378-9.

[FLS+09] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, , and J. Walker. The skein hash function family. 2009. URL: https://www.schneier.com/academic/archives/2009/09/the_skein_hash_funct.html.

[GDM19] Aldo Gunsing, Joan Daemen, and Bart Mennink. Deck-based wide block cipher modes and an exposition of the blinded keyed hashing model. *IACR Transactions on Symmetric Cryptology*, 2019(4):1–22, 2019. URL: https://repository.ubn.ru.nl/bitstream/handle/2066/221667/221667.pdf?sequence=1, doi:10.13154/tosc.v2019.i4.1-22.

[HR03] Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. *Advances in Cryptology – CRYPTO 2003. Lecture Notes in Computer Science*, 2729:482–499, 2003. URL: https://link.springer.com/chapter/10.1007/978-3-540-45146-4_28, doi:10.1007/978-3-540-45146-4_28.

[HR04] Shai Halevi and Phillip Rogaway. A parallelizable enciphering mode. *Topics in Cryptology – CT-RSA 2004. Lecture Notes in Computer Science*, 2964:292–304, 2004. URL: https://link.springer.com/chapter/10.1007/978-3-540-24660-2_23, doi:10.1007/978-3-540-24660-2_23.

[IMPS17] Tetsu Iwata, Kazuhiko Minematsu, Thomas Peyrin, and Yannick Seurin. Zmac: A fast tweakable block cipher mode for highly secure message authentication. *Advances in Cryptology – CRYPTO 2017. Lecture Notes in Computer Science*, 10403:34–65, 2017. URL: https://link.springer.com/chapter/10.1007/978-3-319-63697-9_2, doi:10.1007/978-3-319-63697-9_2.

[JKNS24] Ashwin Jha, Mustafa Khairallah, Mridul Nandi, and Abishanka Saha. Tight security of tnt and beyond. *Advances in Cryptology – EUROCRYPT 2024. Lecture Notes in Computer Science*, 14651:249–279, 2024. URL: https://link.springer.com/chapter/10.1007/978-3-031-58716-0_9, doi:10.1007/978-3-031-58716-0_9.

[JLM+17] Ashwin Jha, Eik List, Kazuhiko Minematsu, Sweta Mishra, and Mridul Nandi.

Xhx – a framework for optimally secure tweakable block ciphers from classical block ciphers and universal hashing. *Progress in Cryptology – LATINCRYPT 2017. Lecture Notes in Computer Science*, 11368:207–227, 2017. URL: https://link.springer.com/chapter/10.1007/978-3-030-25283-0_12, doi:10.1007/978-3-030-25283-0_12.

[JN20] Ashwin Jha and Mridul Nandi. Tight security of cascaded lrw2. *Journal of Cryptology*, 33:1272–1317, 2020. URL: https://link.springer.com/article/10.1007/s00145-020-09347-y, doi:10.1007/s00145-020-09347-y.

[JNP14] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Tweaks and keys for block ciphers: The tweakey framework. *Advances in Cryptology – ASIACRYPT 2014. Lecture Notes in Computer Science*, 8874:274–288, 2014. URL: https://link.springer.com/chapter/10.1007/978-3-662-45608-8_15, doi:10.1007/978-3-662-45608-8_15.

[LL18] ByeongHak Lee and Jooyoung Lee. Tweakable block ciphers secure beyond the birthday bound in the ideal cipher model. *Advances in Cryptology – ASIACRYPT 2018. Lecture Notes in Computer Science*, 11272:305–335, 2018. URL: https://link.springer.com/chapter/10.1007/978-3-030-03326-2_11, doi:10.1007/978-3-030-03326-2_11.

[LMP17] Atul Luykx, Bart Mennink, and Kenneth G. Paterson. Analyzing multi-key security degradation. *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security*, pages 575–605, 2017. URL: https://link.springer.com/chapter/10.1007/978-3-319-70697-9_20#author-information, doi:10.1007/978-3-319-70697-9_20.

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. *Advances in Cryptology – CRYPTO 2002. Lecture Notes in Computer Science*, 2442:31–46, 2002. URL: https://link.springer.com/chapter/10.1007/3-540-45708-9_3, doi:10.1007/3-540-45708-9_3.

[MF04] David A. McGrew and Scott R. Fluhrer. The extended codebook (xcb) mode of operation. *Cryptology ePrint Archive*, 2004. URL: https://eprint.iacr.org/2004/278.

[MI11] Kazuhiko Minematsu and Tetsu Iwata. Building blockcipher from tweakable blockcipher: Extending fse 2009 proposal. *IMA International Conference on Cryptography and Coding. Lecture Notes in Computer Science*, 7089:391–412, 2011. URL: https://link.springer.com/chapter/10.1007/978-3-642-25516-8_24, doi:10.1007/978-3-642-25516-8_24.

[MM07] Kazuhiko Minematsu and Toshiyasu Matsushima. Tweakable enciphering schemes from hash-sum-expansion. *INDOCRYPT 2007. Lecture Notes in Computer Science*, 4859:252–267, 2007. URL: https://link.springer.com/chapter/10.1007/978-3-540-77026-8_19, doi:10.1007/978-3-540-77026-8_19.

[MV05] David A. McGrew and John Viega. The galois/counter mode of operation (gcm). 2005. URL: https://csrc.nist.rip/groups/ST/toolkit/BCM/documents/

proposedmodes/gcm/gcm-revised-spec.pdf.

[Rog04] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes ocb and pmac. *Advances in Cryptology – ASIACRYPT 2004. Lecture Notes in Computer Science*, 3329:16–31, 2004. URL: https://link.springer.com/chapter/10.1007/978-3-540-30539-2_2, doi:10.1007/978-3-540-30539-2_2.

[Ser98] Gadiel Seroussi. Table of low-weight binary irreducible polynomials. 1998. URL: https://shiftleft.com/mirrors/www.hpl.hp.com/techreports/98/HPL-98-135.pdf.

[ST13] Thomas Shrimpton and R. Seth Terashima. A modular framework for building variable-input-length tweakable ciphers. *Advances in Cryptology – ASIACRYPT 2013. Lecture Notes in Computer Science*, 8269:405–423, 2013. URL: https://link.springer.com/chapter/10.1007/978-3-642-42033-7_21, doi:10.1007/978-3-642-42033-7_21.

[WFW05] Peng Wang, Dengguo Feng, and Wenling Wu. Hctr: A variable-input-length enciphering mode. *International Conference on Information Security and Cryptology*, pages 175–188, 2005. URL: https://link.springer.com/chapter/10.1007/11599548_15, doi:10.1007/11599548_15.

[Zha23] Ping Zhang. Universal tweakable even-mansour cipher and its applications. *Frontiers of Computer Science*, 17(174807), 2023. URL: https://link.springer.com/article/10.1007/s11704-022-1466-1, doi:10.1007/s11704-022-1466-1.

Table 4: A summary of ACCOR-HEH benefits compared to the original HCTR.

	HCTR	ACCOR-HEH
Security	$4.5\sigma^2/2^{128}$	Dominated by $0.5\sigma^2/2^{128}$ (when PRF is instantiated with CTR)
Flexibility	Hard-coded primitives and fixed-length tweak.	Varying primitives and variable-length tweak.
Efficiency	Lower security hash primitives scale up (a dominant term of) the security bound linearly.	It is possible to use hash primitives with higher efficiency (see Appendix B).

Appendices

A ACCOR-HEH construction

In this part, we briefly introduce the general construction called ACCOR-HEH, seen in section 3, which follows the Hash-Encrypt-Hash paradigm. It can be thought of as a generalization of HCTR, with two main differences:

1. Variable-length tweak is allowed. This adds greater flexibility as the accordion mode can be used as an AEAD scheme where variable-length associated data acts as (part of) the tweak.

2. The block cipher used to transform the first block (E_K in Figure 1), and the pseudorandom generator (CTR_K), are separated (they are not allowed to share the same key as in HCTR). This brings several benefits:

   - A better security bound: HCTR has a bound of $4.5\sigma^2/2^{128}$, while ACCOR-HEH, with CTR as a pseudorandom function, has a bound of approximately $0.5\sigma^2/2^{128}$. This means the only dominant term in the security bound of ACCOR-HEH comes from the pseudorandom function. In fact, this should be the case as it processes most of the input. Saving a factor of 9 in the security bound triples the life of an accordion key.

   - Again, flexibility: ACCOR-HEH allows plugging in/out different components of the construction easily for different applications. Most parts of the security proof of ACCOR-HEH are reusable.

   - Efficiency: By making the probability of collision after the hash-then-xor part non-dominant in the final security bound, it is possible to plug-in a different, almost-universal hash family with lower security guarantee but higher efficiency.

A summary of ACCOR-HEH benefits is given in Table 4.

A.1 Preliminaries

Distinguishing advantage When an adversary $\mathcal{A}$ tries to distinguish between 2 similar systems SX and SY implementing the same interface IF consisting of the prototypes of functions that $\mathcal{A}$ may invoke, the distinguishing advantage $\texttt{IF}\mathrm{Adv}_{\texttt{SX}}^{\texttt{SY}}(\mathcal{A})$[1] is defined to be $|\Pr[\mathcal{A}^{\texttt{SX}} \to 1] - \Pr[\mathcal{A}^{\texttt{SY}} \to 1]|$. Written this way, SY is usually the ideal version of SX associated with IF. Note that the interface can be omitted if it is clear from context.

In order to denote the maximum distinguishing advantage for a class of resource-limited adversaries, the notation $\mathbf{Adv}_{\texttt{SX}}^{\texttt{SY}}(\mathit{limits})$ is used. Only deterministic adversaries are considered in this case (since a probabilistic adversary's advantage must be bounded by one of some deterministic version with fixed randomness). It is also safe to assume adversaries to use all allowed resources (since they can simply waste unused ones without advantage loss). Note that the limits may also be omitted if they can be deduced from a context.

Block cipher A block cipher BC is used as a pseudorandom permutation in ACCOR-HEH. Let $n, \mathcal{K}_{\texttt{BC}}, \texttt{enc}_{\texttt{BC}}, \texttt{dec}_{\texttt{BC}}$ denote its block size in bits, its key space, its encryption, and its decryption functions, respectively.

BC has good security if it is hard to distinguish from a truly random permutation. This property is represented by $\texttt{SPRP}_n\mathbf{Adv}_{\texttt{BC}}^{\texttt{PERM}_n}(q,t)$, where $\texttt{SPRP}_n$ is an interface allowing an adversary to access a permutation over n-bit strings and its inverse; $\texttt{PERM}_n$ implements $\texttt{SPRP}_n$ by sampling a truly random permutation, while BC (the implementation) initializes BC (the block cipher) with a random key; q, t represent the number of queries made by the adversary and its computational time, respectively.

Stream cipher A stream cipher SC is used as a pseudorandom function in ACCOR-HEH. Let $n_I, n_O, \mathcal{K}_{\texttt{SC}}, \texttt{gen}_{\texttt{SC}}$ denote the sizes in bits of its input and output, its key space, and the output generation function, respectively.

In our case, the security of SC is represented by $\texttt{PRF}_{n_I,n_O}\mathbf{Adv}_{\texttt{SC}}^{\texttt{FUNC}_{n_I,n_O}}(q,\sigma,t)$, where $\texttt{PRF}_{n_I,n_O}$ is an interface allowing an adversary to see the truncated output of a function for some input (exactly n_I-bit long) and output size (up to n_O bits) chosen by the adversary; $\texttt{FUNC}_{n_I,n_O}$ implements $\texttt{PRF}_{n_I,n_O}$ by sampling a truly random function, while SC (the implementation) initializes SC (the stream cipher) with a random key; q, σ, t represent the number of queries, the total (padded) queried blocks $(\sum L_{\text{output}})$[2], and the computational time of the adversary, respectively.

Almost-universal hashing ACCOR-HEH uses an ϵ-almost-universal hash family $\mathcal{H}$. Each member of $\mathcal{H}$ takes as input 2 variable-length bit strings and produces an element of an additively-written group G. It is required that for every tuple $m_1, t_1, m_2, t_2, \Delta$ such that $(m_1, t_1) \neq (m_2, t_2)$ and $\Delta \in G$, we have $\Pr_{h \in \mathcal{H}}[h(m_1, t_1) - h(m_2, t_2) = \Delta] \leq \epsilon$. Note that ϵ may depend on the sizes of the hash inputs.

In ACCOR-HEH, all hash outputs will be later fed into the block cipher, so it is required that $|G| = 2^n$, for example, $\mathbb{F}_{2^n}$ or $\mathbb{Z}_{2^n}$. Converting between an element of G and an n-bit string is implicitly taken.

Transcript and trace A transcript contains communication data (queries and answers) observed on an interface when an adversary interacts with a system. A trace τ extends the transcript to include all internal values of the system. Since only a deterministic adversary

[1]We use this slightly different notation from the literature since it allows us to easily write the distinguishing advantage between two intermediate systems. In the usual notation, e.g., $\mathrm{Adv}_{\texttt{AES}}^{\pm\mathrm{prp}}$, ±prp describes an interface while SY is implied to be the associated ideal implementation.

[2]One can also define σ in term of bits. However, subsequent analyses are only for block-cipher-based stream ciphers, so it is more convenient to work with sizes in blocks.

is considered, each trace is one-to-one corresponding to a tuple of the system's internal coin tossing results. It forks whenever a coin is tossed.

In order to refer to the value of a variable V during the i-th query execution, we use the notation $\tau[i]$.V. When τ is clear from a context, $\mathtt{V}_i$ is used instead. Note that a special variable called func is used to represent the name of the function being invoked during a query execution.

Other notations The following notations will be used for the rest of this section:

- $P, \dot{P}, \bar{P}$: plaintext, first plaintext block and plaintext without first block.
- $C, \dot{C}, \bar{C}$: ciphertext, first ciphertext block and ciphertext without first block.
- T: tweak.
- RAND: an implementation to some interface that always ignores its input and returns truly random bits of appropriate size.
- $|X|, L_X$: length of X in bits and blocks, respectively ($L_X = \lceil |X|/n \rceil$).

A.2 The scheme

The overview of the construction is depicted in Figure 2 while its detailed pseudocode is given in Algorithms 1, 2, 3. The size of plaintext (or ciphertext) is always required to be at least 1 block.

As mentioned earlier, here we have the CTR$_K$ component of HCTR generalized to PRF, which works independently from PRP (the E_K component). We also have a variable-length tweak fed into the hash function, without affecting the overall security of the scheme much.

Algorithm 1 Initialization and sub-functions

```
$h \xleftarrow{\$} \mathcal{H}$

$k_{\mathtt{BC}} \xleftarrow{\$} \mathcal{K}_{\mathtt{BC}}$
function PRP($x$)
    return enc$_{\mathtt{BC}}(k_{\mathtt{BC}}, x)$
function IPRP($y$)
    return dec$_{\mathtt{BC}}(k_{\mathtt{BC}}, y)$

$k_{\mathtt{SC}} \xleftarrow{\$} \mathcal{K}_{\mathtt{SC}}$
function PRF($x$, $l$)
    return gen$_{\mathtt{SC}}(k_{\mathtt{SC}}, x, l)$
```

Algorithm 2 Encryption

```
function ENC($T, P$)
    $P' \leftarrow h(T, \bar{P}) \oplus \dot{P}$
    $C' \leftarrow \mathtt{PRP}(P')$
    $S \leftarrow P' \oplus C'$
    $S' \leftarrow \mathtt{truncate}(S, n_I)$
    $K \leftarrow \mathtt{PRF}(S', |\bar{P}|)$
    $\bar{C} \leftarrow \bar{P} \oplus K$
    $\dot{C} \leftarrow h(T, \bar{C}) \oplus C'$
    return $C$
```

Algorithm 3 Decryption

```
function DEC($T, C$)
    $C' \leftarrow \dot{C} \oplus h(T, \bar{C})$
    $P' \leftarrow \mathtt{IPRP}(C')$
    $S \leftarrow P' \oplus C'$
    $S' \leftarrow \mathtt{truncate}(S, n_I)$
    $K \leftarrow \mathtt{PRF}(S', |\bar{C}|)$
    $\bar{P} \leftarrow \bar{C} \oplus K$
    $\dot{P} \leftarrow P' \oplus h(T, \bar{P})$
    return $P$
```

A.3 Security

Let TVSPRP (tweakable variable-length strong pseudorandom permutation) be the interface which exposes the functions ENC and DEC of ACCOR-HEH, and TVPERM be the interface's ideal implementation which samples a truly random permutation per pair of $T, |P|$ to answer its queries.

To prove the security of `ACCOR-HEH`, $\texttt{TVSPRP}\mathbf{Adv}^{\texttt{TVPERM}}_{\texttt{ACCOR-HEH}}(q,\sigma,t)$ must be given [CDD+24], where q,σ,t refer to the number of queries, the total number of (padded) message blocks ($\sum L_P$), and the computational time of an adversary, respectively.

Since `ACCOR-HEH` and `TVPERM` both act as a set of permutations, a pointless query (a query whose answer is already determined, e.g., repeating a past query) does not give the adversary any extra advantage. As a result, only adversaries that do not make such a type of query are considered.

Distinguishing `RAND` from `TVPERM` To be identical with `TVPERM`, `RAND` needs to output a fresh value per pair of tweak, input-length. So, the most optimal distinguishing attack in this case should be sending a nonce of minimum supported size as message with the same tweak for each query, and waiting until a collision is found. The probability for the success of this attack is bounded by the birthday bound:

$$\mathbf{Adv}^{\texttt{TVPERM}}_{\texttt{RAND}} \leq \frac{q^2}{2} \cdot \frac{1}{2^{|P|_{\min}}} \leq \frac{q^2}{2} \cdot \frac{1}{2^n}$$

Distinguishing $\texttt{ACCOR-HEH}[\texttt{PERM}_n, \texttt{FUNC}_{n_I,n_O}]$ from `RAND` In Algorithm 4, the original implementation of `ACCOR-HEH` initialization and sub-functions in Algorithm 1 is modified. Note that $\mathcal{D}_{\texttt{PRP}}$, $\mathcal{R}_{\texttt{PRP}}$ and $\mathcal{D}_{\texttt{PRF}}$ are dictionaries acting as caches in order to maintain a truly random permutation and a truly random function in a lazy approach. When Algorithm 4 is combined with the unchanged Algorithms 2 and 3, we have the $\texttt{ACCOR-HEH}[\texttt{PERM}_n, \texttt{FUNC}_{n_I,n_O}]$ construction.

One might notice that if the re-assignations of x (or y) at lines 6, 7, 13, 14, 21 are not carried out, we will obtain the $\texttt{ACCOR-HEH}[\texttt{RAND}, \texttt{RAND}]$ construction which is equivalent to a single `RAND`. As a result, the two constructions will be the same as long as the bad events, $B_{\{\text{inP, outP, inF}\}}$ set to 1, never happen. Therefore, in this case, we have:

$$\text{Adv}^{\texttt{RAND}}_{\texttt{ACCOR-HEH}[\texttt{PERM}_n,\texttt{FUNC}_{n_I,n_O}]}(\mathcal{A}) \leq \Pr_{\tau \leftarrow \text{Traces}(\mathcal{A}^{\texttt{RAND}})}[\bigcup_{i=1}^{q}(B_{\text{inP};i} = 1 \cup B_{\text{outP};i} = 1 \cup B_{\text{inF};i} = 1)]$$

Note that the probability is taken over a distribution of traces. We can also have $\tau \leftarrow \text{Traces}(\mathcal{A}^{\texttt{ACCOR-HEH}[\texttt{PERM}_n,\texttt{FUNC}_{n_I,n_O}]})$ since the probability of a good trace is the same in both cases. However, it is more convenient to work with the more ideal system (e.g., trace distribution is uniform for $\mathcal{A}^{\texttt{RAND}}$).

Let an adversary run and interact with `RAND` but interpreted as $\texttt{ACCOR-HEH}[\texttt{RAND}, \texttt{RAND}]$. Suppose that the system is about to execute the i-th query, let us group all possible traces so far by their transcripts. For each group, there should be exactly one trace τ corresponding to one $h \in \mathcal{H}$ (since given any τ, h pair, one can uniquely solve for the outputs of the `RAND` sub-components during the past $i-1$ queries), and the i-th query should already be determined (since only a deterministic adversary is considered). For now, let us fix a group of traces and assume that $\texttt{func}_i = \texttt{ENC}$.

Let us consider $\Pr[B_{\text{inP};i}]$. Since pointless query is not allowed, we have $T_i, P_i \notin \{T_j, P_j\}_{j<i}$. For every index j, there are two cases:

- $T_i, \bar{P}_i = T_j, \bar{P}_j$: This implies $\dot{P}_i \neq \dot{P}_j$, which results in $P'_i \neq P'_j$ ($\Pr[P'_i = P'_j] = 0$).
- $T_i, \bar{P}_i \neq T_j, \bar{P}_j$: The probability $\Pr[P'_i = P'_j] = \Pr_h[h(T_i, \bar{P}_i) \oplus h(T_j, \bar{P}_j) = \bar{P}_j \oplus \bar{P}_i]$ will be bounded by $\epsilon_{i,j}$ because of the almost-universality property of the hash family.

Algorithm 4 $\texttt{ACCOR-HEH}[\texttt{PERM}_n, \texttt{FUNC}_{n_I,n_O}]$ Initialization and sub-functions

```
1: $h \xleftarrow{\$} \mathcal{H}$
2: $\mathcal{D}_{\texttt{PRP}}, \mathcal{R}_{\texttt{PRP}} \leftarrow \{\}, \{\}$
3: function $\texttt{PRP}(x)$
4:     $B_{\text{inP}}, B_{\text{outP}} \leftarrow 0, 0$
5:     $y \xleftarrow{\$} \{0,1\}^n$
6:     if $y \in \mathcal{R}_{\texttt{PRP}}$ then $B_{\text{outP}} \leftarrow 1$; $y \xleftarrow{\$} \{0,1\}^n \setminus \mathcal{R}_{\texttt{PRP}}$     ▷ Bad PRP output
7:     if $x \in \mathcal{D}_{\texttt{PRP}}$ then $B_{\text{inP}} \leftarrow 1$; $y \leftarrow \mathcal{D}_{\texttt{PRP}}[x]$     ▷ Bad PRP input
8:     $\mathcal{D}_{\texttt{PRP}}[x] = y$; $\mathcal{R}_{\texttt{PRP}}[y] = x$;
9:     return $y$
10: function $\texttt{IPRP}(y)$
11:     $B_{\text{inP}}, B_{\text{outP}} \leftarrow 0, 0$
12:     $x \xleftarrow{\$} \{0,1\}^n$
13:     if $x \in \mathcal{D}_{\texttt{PRP}}$ then $B_{\text{outP}} \leftarrow 1$; $x \xleftarrow{\$} \{0,1\}^n \setminus \mathcal{D}_{\texttt{PRP}}$     ▷ Bad IPRP output
14:     if $y \in \mathcal{R}_{\texttt{PRP}}$ then $B_{\text{inP}} \leftarrow 1$; $x \leftarrow \mathcal{R}_{\texttt{PRP}}[y]$     ▷ Bad IPRP input
15:     $\mathcal{D}_{\texttt{PRP}}[x] = y$; $\mathcal{R}_{\texttt{PRP}}[y] = x$;
16:     return $x$
17: $\mathcal{D}_{\texttt{PRF}} \leftarrow \{\}$
18: function $\texttt{PRF}(x, l)$
19:     $B_{\text{inF}} \leftarrow 0$
20:     $y \xleftarrow{\$} \{0,1\}^{n_O}$
21:     if $x \in \mathcal{D}_{\texttt{PRF}}$ then $B_{\text{inF}} \leftarrow 1$; $y \leftarrow \mathcal{D}_{\texttt{PRF}}[x]$     ▷ Bad PRF input
22:     $\mathcal{D}_{\texttt{PRF}}[x] = y$
23:     return $\texttt{truncate}(y, l)$
```

Therefore, $\Pr[B_{\text{inP};i}] \le \sum_{j=1}^{i-1} \epsilon_{i,j}$.

On the other hand, $\Pr[B_{\text{outP};i}] = \Pr[C'_i \in \{C'_j\}_{j<i}] \le (i-1)/2^n$, while $\Pr[B_{\text{inF};i}] = \Pr[S'_i \in \{S'_j\}_{j<i}] \le (i-1)/2^{n_I}$ since C'_i and S'_i have uniform distributions over $\{0,1\}^n$ and $\{0,1\}^{n_I}$, respectively, and there are at most $i-1$ different C'_j or S'_j.

Similar arguments can be applied to any other group of traces with either $\texttt{func}_i = \texttt{ENC}$ or DEC. We conclude that:

$$\begin{aligned}\mathbf{Adv}^{\texttt{RAND}}_{\texttt{ACCOR-HEH}[\texttt{PERM}_n,\texttt{FUNC}_{n_I,n_O}]} &\le \sum_{i=1}^{q}(\sum_{j=1}^{i-1}\epsilon_{i,j} + \frac{i-1}{2^n} + \frac{i-1}{2^{n_I}}) \\ &\le \frac{q^2}{2}(\frac{1}{2^n} + \frac{1}{2^{n_I}}) + \sum_{1\le j<i\le q}\epsilon_{i,j}\end{aligned}$$

Distinguishing $\texttt{ACCOR-HEH}[\texttt{PERM}_n, \texttt{SC}]$ from $\texttt{ACCOR-HEH}[\texttt{PERM}_n, \texttt{FUNC}_{n_I,n_O}]$ Given any distinguisher for these two systems, one can always derive a distinguisher for SC and $\texttt{FUNC}_{n_I,n_O}$ with the same advantage by executing in-the-head the logic of ACCOR-HEH excluding the PRF part. As a result:

$$\mathbf{Adv}^{\texttt{ACCOR-HEH}[\texttt{PERM}_n,\texttt{FUNC}_{n_I,n_O}]}_{\texttt{ACCOR-HEH}[\texttt{PERM}_n,\texttt{SC}]}(q,\sigma,t) \le \texttt{PRF}\mathbf{Adv}^{\texttt{FUNC}_{n_I,n_O}}_{\texttt{SC}}(q,\sigma-q,t+O(\sigma))$$

Where $O(\sigma)$ refers to the time spent on executing the shared steps of the two ACCOR-HEH systems, provided that the cost for processing the tweaks is insignificant compared to the messages.

Distinguishing $\texttt{ACCOR-HEH}[\texttt{BC}, \texttt{SC}]$ from $\texttt{ACCOR-HEH}[\texttt{PERM}_n, \texttt{SC}]$ With a similar argument, we have:

$$\mathbf{Adv}^{\texttt{ACCOR-HEH}[\texttt{PERM}_n,\texttt{SC}]}_{\texttt{ACCOR-HEH}[\texttt{BC},\texttt{SC}]}(q,\sigma,t) \le \texttt{SPRP}\mathbf{Adv}^{\texttt{PERM}_n}_{\texttt{BC}}(q,t+O(\sigma))$$

Overview Finally, by summing up all the above bounds, we have:

$$\begin{aligned}\mathbf{Adv}^{\texttt{TVPERM}}_{\texttt{ACCOR-HEH}}(q,\sigma,t) \le\ & \frac{q^2}{2}(\frac{2}{2^n} + \frac{1}{2^{n_I}}) + \sum_{1\le j<i\le q}\epsilon_{i,j} \\ &+ \texttt{PRF}\mathbf{Adv}^{\texttt{FUNC}_{n_I,n_O}}_{\texttt{SC}}(q,\sigma-q,t+O(\sigma)) \\ &+ \texttt{SPRP}\mathbf{Adv}^{\texttt{PERM}_n}_{\texttt{BC}}(q,t+O(\sigma))\end{aligned} \tag{5}$$

The security of ACCOR-HEH has been reduced to its underlying primitives.

In order to demonstrate the flexibility of ACCOR-HEH, in addtition to the constructions in the main sections, two more approaches are given.

B ACCOR-S'

This section describes a concrete instantiation of ACCOR-HEH called ACCOR-S'. The primitives are instantiated as follows.

Block cipher AES-256 is used. The block size n is 128 bits.

Stream cipher CTR[AES-256] is used. Let the counter size c be equal to 32 bits, and the PRF input size n_I be equal to $n - c = 96$ bits. We have the maximum PRF output size, $n_O = 2^c n = 2^{39}$ bits, which corresponds with 64GB.

The security bound of CTR is known to be the birthday bound:

$$\mathtt{PRF}\mathbf{Adv}^{\mathtt{FUNC}_{n_I,n_O}}_{\mathtt{CTR}[\mathrm{AES}_{256}]}(q,\sigma,t) \le \frac{\sigma^2}{2^{129}} + \mathtt{PRP}\mathbf{Adv}^{\mathtt{PERM}_{128}}_{\mathrm{AES}_{256}}(\sigma, t + O(\sigma))$$

Almost universal hash POLY1305 [Ber05] is used. Since POLY1305 takes a sequence of 128-bit blocks as input while h described in the generic construction has two arguments, an unambiguous encoding scheme is needed to convert two bitstrings into a single block sequence. Inspired by GCM's GHASH [MV05], in `ACCOR-S'`, the following encoding format is used: padded tweak (L_T blocks), padded message without first block ($L_P - 1$ blocks), tweak length (half block), message length (half block). This results in a sequence of $L_T + L_P$ blocks.

For POLY1305, we have:

$$\begin{aligned}\epsilon_{i,j} &= \frac{\max(L_{\mathrm{input}_i}, L_{\mathrm{input}_j})}{2^{103}} = \frac{\max(L_{T_i} + L_{P_i}, L_{T_j} + L_{P_j})}{2^{103}} \\ &\le \frac{L_{T;\max} + L_{P_i} + L_{P_j}}{2^{103}}\end{aligned}$$

Therefore:

$$\begin{aligned}\sum_{1\le j<i\le q} \epsilon_{i,j} &\le \frac{q^2 L_{T;\max}}{2^{104}} + \frac{\sum_{1\le j<i\le q}(L_{P_i} + L_{P_j})}{2^{103}} \\ &\le \frac{q^2 L_{T;\max}}{2^{104}} + \frac{q\sigma}{2^{103}}\end{aligned}$$

Note that the last inequality is due to each of $L_{P_1}, L_{P_2}, \ldots$ appearing less than q times in the sum.

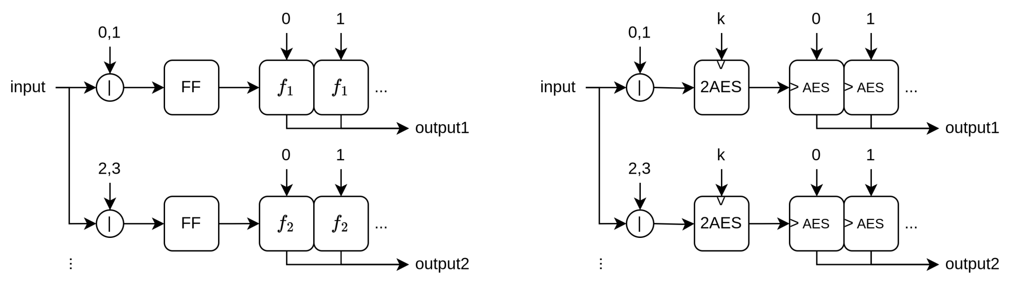
Overview Substituting these into 5, we have:

$$\begin{aligned}\mathbf{Adv}^{\mathtt{TVPERM}}_{\mathtt{ACCOR\text{-}S'}}(q,\sigma,t) \le{}& \frac{q^2}{2}\left(\frac{2}{2^{128}} + \frac{1}{2^{96}}\right) + \frac{q^2 L_{T;\max}}{2^{104}} + \frac{q\sigma}{2^{103}} \\ &+ \frac{\sigma^2}{2^{129}} + \mathtt{PRP}\mathbf{Adv}^{\mathtt{PERM}_{128}}_{\mathrm{AES}_{256}}(\sigma - q, t + O(\sigma)) \\ &+ \mathtt{SPRP}\mathbf{Adv}^{\mathtt{PERM}_{128}}_{\mathrm{AES}_{256}}(q, t + O(\sigma))\end{aligned}$$

For a concrete example, let $q \le 2^{24}$ (about 16 million queries), $\sigma \le 2^{48}$ (about 9000 TB of total data), and $L_{T;\max} \le 2^{24}$ (about 537MB of maximum tweak size). The probability that an adversary can break `ACCOR-S'` is bounded by $2^{-30.19}$, provided that AES-256 is indistinguishable from a random permutation.

C `ACCOR-L'`

This section attempts to achieve BBB security, i.e., having σ larger than 2^n while keeping the distinguishing advantage insignificant. It is important to note that if we keep obtaining randomness from a single permutation, the birthday bound can not be avoided. Therefore, block cipher re-keying must happen inside the `PRF` of `ACCOR-HEH` as it processes most of the input.



.

Figure 5: The ideal and actual implementations of the PRF of ACCOR-L'. In the ideal implementation, FF refers to a function factory that samples and returns a random function for each unique FF input. In the actual implementation, an AES instantiated with a random key (obtained from encrypting the FF input) is returned instead. Note that 2AES means 2 parallel AES while $\mathit{input}|\{0,1\} = \{\mathit{input}|0, \mathit{input}|1\}$

Block cipher ACCOR-L' also uses AES-256 as block cipher.

Stream cipher ACCOR-L' uses a custom construction in order to enforce that no single permutation is used to draw more than l blocks of randomness. Inside the PRF, it uses outputs from CTR as keys (two 128-bit CTR output blocks for one 256-bit key) for instantiation of AES whenever necessary, and constants $(0, 1, 2, ..., l)$ as inputs to these instantiated ciphers for the actual output randomness (see Figure 5). The security now relies on the multi-key security of the block cipher (it is hard to distinguish a set of block ciphers instantiated with independent random keys from a set of truly random permutations) [LMP17]. Omitting the advantage related to distinguishing (set of) AES from (set of) random permutation, the advantage bound for this PRF is expected to be:

$$\frac{(2f)^2}{2^{129}} + \frac{fl^2}{2^{129}},$$

where f denotes the number of calls to FF. Note that both terms are caused by using permutations as sources of randomness. $(2f)^2/2^{129}$ refers to the $2f$ calls to AES to produce random keys, while $fl^2/2^{129}$ refers to l calls to each of f instantiated permutations.

Almost universal hash ACCOR-L' uses GHASH [MV05], which has a similar behaviour to POLY1305 except for a constant factor of 2^{25} reduction to ϵ. The encoding scheme is kept unchanged (in fact, GHASH is already defined to accept two inputs). With similar arguments in the ACCOR-S' section, we have in this case:

$$\sum_{1 \le j < i \le q} \epsilon_{i,j} \le \frac{q^2 L_{T;\max}}{2^{129}} + \frac{q\sigma}{2^{128}}$$

Overview Substituting these into 5 while dropping all bounds related to distinguishing AES from a random permutation, we have a bound for the security gap between breaking ACCOR-L' and breaking AES:

$$\frac{q^2}{2}\left(\frac{2}{2^{128}} + \frac{1}{2^{96}}\right) + \frac{q^2 L_{T;\max}}{2^{129}} + \frac{q\sigma}{2^{128}} + \frac{(2f)^2}{2^{129}} + \frac{fl^2}{2^{129}}$$

Note that $f = \sum_{i=1}^{q} \lceil (L_{P_i} - 1)/l \rceil < \sum_{i=1}^{q} (L_{P_i}/l + 1) = \sigma/l + q$.

For a concrete example, let $q \leq 2^{30}$ (around one billion queries), $\sigma \leq 2^{70}$ $(37.78ZB)$, $L_{T;\max} \leq 2^{30}$ $(34.36GB)$, and $l = 2^{30}$, the probability that an adversary can break `ACCOR-L'` is bounded by, approximately, $2^{-27.41}$, provided that AES-256 is indistinguishable from a random permutation.