HPolyC: length-preserving encryption for entry-level processors

Paul Crowley and Eric Biggers

Google LLC
{paulcrowley,ebiggers}@google.com

August 1, 2018

Abstract

We present HPolyC, a construction which builds on Poly1305, XChaCha12, and a single block cipher invocation per message to offer length-preserving encryption with a fast constant-time implementation where crypto instructions are absent. On an ARM Cortex-A7 processor, HPolyC decrypts 4096-byte messages at 14.5 cycles per byte, over four times faster than AES-256-XTS. Assuming secure primitives, we prove an advantage bound of $\approx 2^{-111}q^2(l+156)$, where q is the number of queries and l is the sum of message and tweak length in bits.

1 Introduction

Two aspects of disk encryption make it a challenge for cryptography. First, performance is critical; every extra cycle is a worse user experience, and on a mobile device a reduced battery life. Second, the ciphertext can be no larger than the plaintext: a sector-sized read or write to the filesystem must mean a sector-sized read or write to the underlying device, or performance will again suffer greatly (as well as, in the case of writes to flash memory, the life of the device). Nonce reuse is inevitable as there is nowhere to store a varying nonce, and there is no space for a MAC; thus standard constructions like AES-GCM are not an option and standard notions of semantic security are unachievable. The best that can be done under the circumstances is a "tweakable super-pseudorandom permutation": an attacker with access to both encryption and decryption functions who can choose tweak and plaintext/ciphertext freely is unable to distinguish it from a family of independent random permutations.

1.1 History

Hasty Pudding Cipher [Sch98] was a variable-input-length primitive presented to the AES contest. A key innovation was the idea of a "spice", which was later formalized as a "tweak" in [LRW02]. Another tweakable large-block primitive was Mercy [Cro01], cryptanalyzed in [Flu02].

[LR88] (see also [Mau93; Pat91]) shows how to construct a pseudorandom permutation using a three-round Feistel network of pseudorandom functions; proves that this is not a secure super-pseudorandom permutation (where the adversary has access to decryption as well as encryption) and that four rounds suffice for this aim. BEAR and LION [AB96] apply this result to an unbalanced Feistel network to build a large-block cipher from a hash function and a stream cipher (see also BEAST [Luc96a]).

[Luc96b] shows that a universal function (here called a "difference concentrator") suffices for the first round, which [NR99] extends to four-round function to build a super-pseudorandom permutation.

More recently, proposals in this space have focused on the use of block ciphers. VIL mode [BR99] is a CBC-MAC based two-pass variable-input-length construction which is a PRP but not an SPRP. CMC mode [HR03] is a true SPRP using two passes of the block cipher; EME mode [HR04] is similar but parallelizable, while EME* mode [Hal05] extends EME mode to handle blocks that are not a multiple of the block cipher size. PEP [CS06], TET [Hal07], and HEH [Sar07] have a mixing layer either side of an ECB layer.

XCB [MF07] is a block-cipher based unbalanced three-round Feistel network with an ϵ-almost-XOR-universal hash function for the first and third rounds ("hash-XOR-hash"), which uses block cipher invocations on the narrow side of the network to ensure that the network is an SPRP, rather than just a PRP; it also introduces a tweak. HCTR [WFW05; CN08], HCH [CS08], and HMC [Nan08] reduce this to a single block cipher invocation within the Feistel network. These proposals require either two AES invocations, or an AES invocation and two $\mathrm{GF}(2^{128})$ multiplications, per 128 bits of input.

1.2 Our contribution

On the ARM architecture, the ARMv8 Cryptography Extensions include instructions that make AES and $\mathrm{GF}(2^{128})$ multiplications much more efficient. However, smartphones designed for developing markets often use lower-end processors which don't support these extensions, and as a result there is no existing SPRP construction which performs acceptably on them.

On such platforms stream ciphers such as ChaCha12 [Ber08a] significantly outperform block ciphers in cycles per byte, especially with constant-time implementations. Similarly, absent specific processor support, Poly1305 hash [Ber05b] will be much faster than a $\mathrm{GF}(2^{128})$ polynomial hash. Since these are the operations that act on the bulk of the data in a disk-sector-sized block, a hash-XOR-hash mode of operation relying on them should achieve much improved performance on such platforms.

To this end, we present HPolyC, which:

- is a tweakable, variable-input-length, super-pseudorandom permutation
- has a security bound quadratic in the number of queries and linear in message length
- is highly key agile
- is highly parallelizable
- needs only three passes over the bulk of the data, or two if the XOR is combined with the second hash.

Without special cases or extra setup, HPolyC handles:

- messages that are not a multiple of 128 bits in length
- tweak lengths from 0 to $2^{32}-1$ bits
- varying message and tweak lengths for the same keys.

The proof of security differs from other hash-XOR-hash modes in three ways. First, Poly1305 hash is not XOR universal, but universal over $\mathbb{Z}/2^{128}\mathbb{Z}$, so for XOR of hash values we substitute addition and subtraction in this group. Second, using the XSalsa20 construction [Ber11], we can directly build a stream cipher which takes a 192-bit nonce to generate a 2^{73}-bit stream, simplifying the second Feistel operation and associated proof, as well as subkey generation. Finally, Poly1305 hash has a much weaker security bound than the $\mathrm{GF}(2^{128})$ polynomial hash; the proof is shaped around ensuring we pay the smallest multiple of this cost we can.

2 Specification

HPolyC divides the input into a right-hand block of $n = 128$ bits and a left-hand block with the remainder of the input, and uses a stream cipher S and an ϵ-almost-Δ-universal function H to build an unbalanced Feistel network that includes one invocation of a block cipher E.

We derive ciphertext C from plaintext P and tweak T as shown in Figure 2. $||$ represents concatenation, and P_R, P_M, C_M, C_R are n bits long. encode is an injective encoding function defined below. Byte/bit mapping is little endian, as is all arithmetic. $\boxplus$ represents addition mod 2^n, and $\boxminus$ subtraction. $X \leftarrow\!\!\oplus Y$ is the bitwise XOR of X with the first $|X|$ bits of Y. Partial application is implicit; if we define $f : A \times B \to C$ and $a \in A$ then $f_a : B \to C$ and if f_a^{-1} exists then $f_a^{-1}(f_a(b)) = b$.

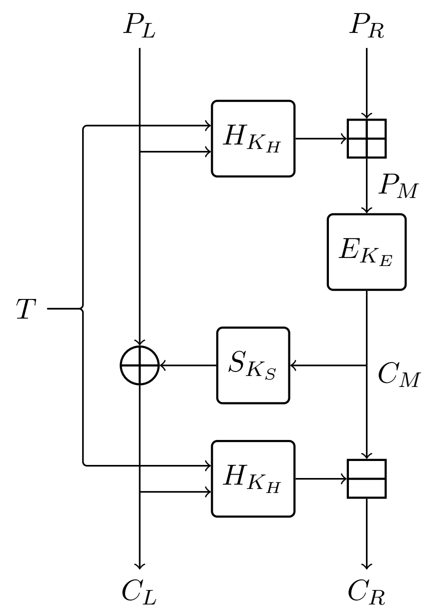

Figure 1: HPolyC

procedure $\textsc{HPolyCEncrypt}(T, P)$
  $P_L || P_R \leftarrow P$
  $P_M \leftarrow P_R \boxplus H_{K_H}(\text{encode}(T, P_L))$
  $C_M \leftarrow E_{K_E}(P_M)$
  $C_L \leftarrow P_L \leftarrow\!\!\oplus S_{K_S}(C_M)$
  $C_R \leftarrow C_M \boxminus H_{K_H}(\text{encode}(T, C_L))$
  $C \leftarrow C_L || C_R$
  return C
end procedure
procedure $\textsc{HPolyCDecrypt}(T, C)$
  $C_L || C_R \leftarrow C$
  $C_M \leftarrow C_R \boxplus H_{K_H}(\text{encode}(T, C_L))$
  $P_L \leftarrow C_L \leftarrow\!\!\oplus S_{K_S}(C_M)$
  $P_M \leftarrow E^{-1}_{K_E}(C_M)$
  $P_R \leftarrow P_M \boxminus H_{K_H}(\text{encode}(T, P_L))$
  $P \leftarrow P_L || P_R$
  return P
end procedure

Figure 2: Pseudocode for HPolyC

2.1 Hash

Poly1305 [Ber05b] is a MAC which combines the use of AES with an ϵ-almost-Δ-universal (ϵAΔU) polynomial hash function. RFC 7539 [NL15] takes this ϵAΔU polynomial function and uses it without AES to build an AEAD mode based on ChaCha20. Here we call this ϵAΔU function $H : \mathcal{K}_H \times \{0,1\}^* \rightarrow \{0,1\}^n$ where $\mathcal{K}_H$ is the 2^{106}-bit keyspace; it is this function, rather than the Poly1305 MAC itself, that we use in HPolyC. Many Poly1305 libraries take parameters $K_H || g, m$ and return $g \boxplus H_{K_H}(m)$; where subtraction is needed we use bitwise inversion and the identity $g \boxminus g' = \neg((\neg g) \boxplus g')$.

The key for this function is required to have certain bits clear per the "Keys" subsection of the specification [Ber05b]; the sample implementation [Ber05a] and [NL15] call clearing these bits "clamping", and following those specifications we define Poly1305Clamp below, with $\wedge$ representing bitwise AND.

$$\text{Poly1305Clamp}(\overline{K_H}) = \overline{K_H} \wedge 1^{28}||0^6||1^{26}||0^6||1^{26}||0^6||1^{26}||0^4$$

2.2 Encoding function

We hash the tweak and the left side of the message, so we need an injective encoding function. Poly1305 hash's ϵAΔU guarantee holds for messages of differing lengths, so we only need to length-encode one of the two parameters.

$[|T|]_{32}$ is a 32-bit little endian encoding of the length of the tweak in bits; we require that $0 \leq |T| < 2^{32}$. v is the least integer ≥ 0 such that n divides $32 + |T| + v$, and we define:

$$\begin{aligned} \text{encode} : \{0,1\}^* \times \{0,1\}^* &\rightarrow \{0,1\}^* \\ \text{encode}(T, L) &= [|T|]_{32} || T || 0^v || L \end{aligned}$$

2.3 Block cipher

The 128-bit block cipher $E : \mathcal{K}_E \times \{0,1\}^n \rightarrow \{0,1\}^n$ is only invoked once no matter the size of the input, so for disk sector-sized inputs its performance isn't critical; we have tested AES-256 [NIS01].

2.4 Stream cipher

The ChaCha12 stream cipher [Ber08a] defines a PRF which takes a 64-bit (or 96-bit [NL15]) nonce and an integer stream offset and returns a 512-bit output, and concatenates successive outputs to define a function from key and nonce to stream in a seekable way. Since C_M is larger than the ChaCha12 nonce size, we use the XSalsa20 construction [Ber11] initially proposed for Salsa20 [Ber08b; Ber06] to construct XChaCha12 which extends the nonce to 192 bits (as libsodium [Den17] does with ChaCha20), and pad with a 1 followed by zeroes. For a given key and nonce, XChaCha12 produces $l_S = 2^{73}$ bits of output; we therefore require that the HPolyC plaintext length be within the bounds $n \leq |P| \leq l_S + n$.

$$\begin{aligned} S : &\mathcal{K}_S \times \{0,1\}^n \rightarrow \{0,1\}^{l_S} \\ S_{K_S}(C_M) = &\text{XChaCha12}_{K_S}(C_M || 1 || 0^{63}) \end{aligned}$$

2.5 Key derivation

HPolyC takes a 256-bit key K_S, and derives keys K_H and K_E using XChaCha12:

$$\begin{aligned} \overline{K_H} || K_E || \ldots &= \text{XChaCha12}_{K_S}(1 || 0^{191}) \\ K_H &= \text{Poly1305Clamp}(\overline{K_H}) \end{aligned}$$

where $|\overline{K_H}| = 128$ and $|K_E| = 256$. Note that the nonce used here is distinct from all nonces used for S_{K_S}.

3 Design

Any secure PRP must have a pass that reads all of the plaintext, followed by a pass that modifies it all. A secure SPRP must have the same property in the reverse direction; a three-pass structure therefore seems natural. ϵAΔU functions are the fastest options for reading the plaintext in a cryptographically useful way, and stream ciphers are the fastest options for modifying it. ϵAΔUs are typically much faster than stream ciphers, and so the hash-XOR-hash structure emerges as the best option for performance. This structure also has the advantage that it naturally handles blocks in non-round sizes; many large-block modes need extra wrinkles akin to ciphertext stealing to handle the case where the large-block size is not a multiple of the block size of the underlying primitive.

[LR88] observes that a three-round Feistel network cannot by itself be a secure SPRP; a simple attack with two plaintexts and one ciphertext distinguishes it. A single block cipher call in the narrow part of the unbalanced network suffices to frustrate this attack; the larger the block, the smaller the relative cost of this call. Compared to HCTR [WFW05] or HCH [CS08], we sacrifice symmetry of encryption with decryption in return for the ability to run the block cipher and stream cipher in parallel when decrypting. For disk encryption, decryption performance matters most: reads are more frequent than writes, and reads generally affect user-perceived latency, while operating systems can usually perform writes asynchronously in the background.

It's unusual for a construction to require three distinct building blocks. More commonly, a hash-XOR-hash mode will use a block cipher both on the narrow side of the block, and to build the stream cipher in the XOR phase (eg using CTR mode [LWR00]). Using XChaCha12 in place of a block cipher affords a significant increase in performance; however it cannot easily be substituted in the narrow side of the cipher. [Sar09; Sar11; CMS13; Cha+17] use only an ϵAXU function and a stream cipher, and build a hash-XOR-hash SPRP with a construction that uses a four-round Feistel network over the non-bulk side of the data broken into two halves. However if we were to build this using XChaCha12, such a construction would require four extra invocations of ChaCha per block, which would be a much greater cost than one block cipher invocation.

The advantage bound we are able to prove is limited mainly by the 106-bit keyspace of the Poly1305 hash. [Ber05b] specifies that 22 bits of the 128-bit key must be zero, to facilitate fast implementation. Many modern implementations do not make use of this, and would work equally well with keys that were not zeroed in this way, which could improve the advantage bound by almost a factor of 2^{-22}. However the current advantage bound is tight enough for our purposes.

HPolyC does not consider an attack model in which derived keys are presented as input. A trivial distinguisher would be to encrypt $0^{(k+1)n}||K_H$ and $0^{kn}||1||0^{2n-1}$ with the same tweak for some k; the resulting ciphertexts will share the same kn-bit prefix. Length-preserving encryption which is KDM-secure in the sense of [BRS03] is impossible, since it is trivial for the attacker to submit a query with a g-function that constructs a plaintext whose ciphertext is all zeroes. Whether there is a notion of KDM-security that can be applied in this domain is an open problem. Users must take care to protect the keys from being included in the input.

For a new key, it is necessary to compute the keys for hash and the block cipher, and schedule the block cipher key; if the hash is to be calculated in parallel, it can also be useful to cache powers of the hash key. ChaCha12 has no key schedule and makes no use of precomputation; XChaCha12 has a "nonce scheduling" step that must be called once to compute subkeys and once for each HPolyC encryption or decryption. No extra work needs to be done for differing message or tweak lengths, unless new powers of the hash key are needed for parallelism.

For storage encryption, tweaks will often be short and fixed-length. With a tweak of 96 bits or fewer, only a single $\mathbb{Z}/(2^{130}-5)\mathbb{Z}$ multiplication (the core operation of Poly1305) is needed before processing the message. We tolerate the extra complexity of padding in our encode function so that plaintext/ciphertext is aligned to the Poly1305 block size. If the tweak is long, the partial result of hashing it can be used both for encoding and decoding; an earlier version of HPolyC included a "disambiguation bit" in the input to the hash, but closer analysis showed this to be unnecessary.

Poly1305 and ChaCha12 are both designed such that the most natural fast implementations are constant-time and free from data-dependent lookups. So long as the block cipher implementation also has these properties, HPolyC will inherit security against this class of side-channel attacks.

Both Poly1305 and ChaCha12 are highly parallelizable. The stream cipher and second hash stages can also be run in combination for a total of two passes over the bulk of the data, unlike a mode such as HEH [Sar07] which requires at least three. We put the "special" block on the right so that in typical uses the bulk encryption has the best alignment for fast operations.

4 Performance

In Table 1 we show performance on an ARM Cortex-A7 processor in the Snapdragon 2100 chipset running at 1.094 GHz. This processor supports the NEON vector instruction set, but not the ARM cryptographic extensions; it is used in many smartphones and smartwatches, especially low-end or older devices, and is representative of the kind of platform we mean to target. Where the figures are the same, a single row is shown for both encryption and decryption.

We have prioritized performance on 4096-byte messages, but we also tested

Algorithm	Cycles per byte (4096-byte sectors)	Cycles per byte (512-byte sectors)
Poly1305	2.94	3.29
ChaCha8	5.43	5.55
ChaCha12	7.58	7.72
ChaCha20	11.9	12.0
HPolyC-XChaCha8-AES (encryption)	12.1	18.4
HPolyC-XChaCha8-AES (decryption)	12.3	19.8
HPolyC-XChaCha12-AES (encryption)	**14.3**	**21.1**
HPolyC-XChaCha12-AES (decryption)	**14.5**	**22.5**
Speck128/256-XTS	15.9	16.9
HPolyC-XChaCha20-AES (encryption)	18.8	26.5
HPolyC-XChaCha20-AES (decryption)	18.9	28.0
NOEKEON-XTS	27.0	27.8
XTEA-XTS	28.7	29.7
AES-128-XTS (encryption)	36.3	38.9
AES-128-XTS (decryption)	43.1	45.6
AES-256-XTS (encryption)	49.4	52.9
AES-256-XTS (decryption)	59.0	62.5

Table 1: Performance on ARM Cortex-A7

512-byte messages. 512-byte disk sectors were the standard until the introduction of Advanced Format in 2010; modern large hard drives and flash drives now use 4096-byte sectors. On Linux, 4096 bytes is the standard page size, the standard allocation unit size for filesystems, and the granularity of *fscrypt* file-based encryption, while *dm-crypt* full-disk encryption has recently been updated to support this size.

For comparison we evaluate against various block ciphers in XTS mode [IEE08]: AES [NIS01], Speck [Bea+13; Bea+15; Bea+17], NOEKEON [Dae+00], and XTEA [NW97]. We also include the performance of ChaCha and Poly1305 by themselves for reference. We used the fastest constant-time implementation of each algorithm we were able to find or write for the platform; see Table 2. In every case except `aes_ti.c`, the performance-critical parts were written in assembly language using NEON instructions.

HPolyC is the only algorithm in Table 1 which is a tweakable super-pseudorandom permutation over the entire sector. We expect any AES-based construction to that end to be significantly slower than AES-XTS. HPolyC has a larger per-message overhead than XTS; both require one extra block cipher invocation per message, but HPolyC must also perform one extra ChaCha permutation for the XChaCha construction and one extra Poly1305 block for the tweak.

We conclude that for 4096-byte sectors, HPolyC-XChaCha12-AES can perform as well as an aggressively designed block cipher (Speck128/256) in XTS mode.

Algorithm	Source	Notes
ChaCha	Linux v4.17	`chacha20-neon-core.S`, modified to support ChaCha8 and ChaCha12
Poly1305	OpenSSL 1.1.0h	`poly1305-armv4.S`, modified to allow key powers to be computed just once per key
AES	Linux v4.17	`aes_ti.c`, used once per message in HPolyC
AES-XTS	Linux v4.17	`aes-neonbs-core.S` (bit-sliced)
Speck128/256-XTS	Linux v4.17	`speck-neon-core.S`
NOEKEON-XTS	ours	
XTEA-XTS	ours	

Table 2: Implementations

Efficient implementations of Poly1305 and ChaCha are available for many platforms, as these algorithms are well-suited for implementation with either general-purpose scalar instructions or with general-purpose vector instructions such as NEON or AVX2. For a greater margin of security at a slower speed, ChaCha20 can be used instead of ChaCha12; the same stream cipher must be used for key derivation as for the Feistel function.

5 Security reduction

HPolyC is a tweakable, variable-input-length, secure pseudorandom permutation: an attacker succeeds if they distinguish it from a family of independent random permutations indexed by input length and tweak, given access to both encryption and decryption oracles.

Given keys K_S, K_H, K_E, HPolyC is the conjugation of an inner transform by an outer:

$$\begin{aligned}
\mathrm{HPolyC} : \{0,1\}^* \times \{0,1\}^l \times \{0,1\}^n &\to \{0,1\}^l \times \{0,1\}^n \\
\mathrm{HPolyC}_T &= \phi^{-1}_{K_H,T} \circ \theta_{E_{K_E}, S_{K_S}} \circ \phi_{K_H,T} \\
\theta_{e,s}(P_L, P_M) &= (P_L \hookleftarrow\!\oplus\, s(e(P_M)), e(P_M)) \\
\phi_{K_H,T}(L, R) &= (L, R \boxplus H_{K_H}(\mathrm{encode}(T, L)) \\
\phi^{-1}_{K_H,T}(L, R) &= (L, R \boxminus H_{K_H}(\mathrm{encode}(T, L))
\end{aligned}$$

These are families of length-preserving functions parameterized by the length $|P_L| = |L| = |C_L| = l \in \mathbb{N}$; for notational convenience we leave this parameter implicit.

We prove a security bound for HPolyC in three stages:

- we consider distinguishers for the inner construction θ in an attack model which forbids "inner collisions" in queries
- we prove a bound on the probability of an attacker causing an inner collision
- we put this together to bound the success probability of a distinguisher against HPolyC.

At each stage, we consider an attacker $\mathcal{A}^{\mathcal{E},\mathcal{D}}$ who makes q queries to oracles for two length-preserving function families which take a tweak; the attacker is always free to vary the length of input and tweak.

$$\begin{aligned}\mathcal{E},\mathcal{D} : \{0,1\}^* \times \{0,1\}^l \times \{0,1\}^n &\rightarrow \{0,1\}^l \times \{0,1\}^n\\ (C_L, C_R) &\leftarrow \mathcal{E}_T(P_L, P_R)\\ (P_L, P_R) &\leftarrow \mathcal{D}_T(C_L, C_R)\end{aligned}$$

5.1 Inner part

We consider an attacker trying to distinguish an idealized θ from a pair of families of random length-preserving functions, but we forbid the attacker from causing "inner collisions": having made either of the queries:

- $(C_L, C_M) \leftarrow \mathcal{E}_T(P_L, P_M)$
- $(P_L, P_M) \leftarrow \mathcal{D}_T(C_L, C_M)$

both of the queries below are subsequently disallowed:

- $\mathcal{E}.(\cdot\,, P_M)$
- $\mathcal{D}.(\cdot\,, C_M)$

where $\cdot$ represents any value. However assuming $C_M \neq P_M$, this does not disallow subsequent queries of the form $\mathcal{D}.(\cdot\,, P_M)$ or $\mathcal{E}.(\cdot\,, C_M)$ and it's important to consider such queries at each step below. We write P_M/C_M for the second argument and result to match notation used for θ elsewhere. At this stage, we consider computationally unbounded attackers; we'll consider resource-bounded attackers in subsection 5.3.

In what follows we use a standard concrete security hybrid argument per [Bel+97; Sho04]: for a fixed class of attacker, distinguishing advantage obeys the triangle inequality and forms a pseudometric space. We consider a sequence of experiments, bound the distingushing advantage between successive experiments, and thereby prove an advantage bound for a distinguisher between the first and the last experiment which is the sum of the advantage bound between each successive experiment.

5.1-randinner: $\mathcal{E}$ and $\mathcal{D}$ are families of random functions. Since by our constraints above every query to each is distinct, every output of the appropriate length is equally likely. Wherever we specify that an experiment uses multiple random functions, those functions are independent unless stated otherwise.

5.1-notweak: Ignore the tweak: let $\overline{\mathcal{E}}, \overline{\mathcal{D}}$ be random length-preserving function families from $\{0,1\}^l \times \{0,1\}^n \to \{0,1\}^l \times \{0,1\}^n$ and let $\mathcal{E}_T = \overline{\mathcal{E}}$ and $\mathcal{D}_T = \overline{\mathcal{D}}$ for all T. Since by the above constraints, for each random function the second argument in every query is still always distinct, every output of the appropriate length is equally likely as before, and this is indistinguishable from 5.1-randinner.

5.1-doublerf: Use a Feistel network in which the stream cipher nonce includes both P_M and C_M.

- $F_S : \{0,1\}^{2n} \to \{0,1\}^{l_S}$ is a random function
- $F_E, F_D : \{0,1\}^n \to \{0,1\}^n$ are random functions
- $\mathcal{E}_T(P_L, P_M) = (P_L \leftarrow\!\oplus F_S(P_M || F_E(P_M)), F_E(P_M))$
- $\mathcal{D}_T(C_L, C_M) = (C_L \leftarrow\!\oplus F_S(F_D(C_M) || C_M), F_D(C_M))$

Again, the constraints above ensure that for each random function, every query is distinct, every output of the appropriate length is equally likely as before, and this is indistinguishable from 5.1-notweak and 5.1-randinner.

5.1-rpswitch: Substitute a random permutation for the pair of random functions.

- $F_S : \{0,1\}^{2n} \to \{0,1\}^{l_S}$ is a random function as before
- $\pi : \{0,1\}^n \to \{0,1\}^n$ is a random permutation
- $\mathcal{E}_T(P_L, P_M) = (P_L \leftarrow\!\oplus F_S(P_M || \pi(P_M)), \pi(P_M))$
- $\mathcal{D}_T(C_L, C_M) = (C_L \leftarrow\!\oplus F_S(\pi^{-1}(C_M) || C_M), \pi^{-1}(C_M))$ ie $\mathcal{D}_T = \mathcal{E}_T^{-1}$

The constraints above rule out "pointless" queries on π, so per section C of [HR03] the advantage in distinguishing this from 5.1-doublerf is at most $2^{-n}\binom{q}{2}$.

5.1-halfrf: Replace the two-argument F_S with a single-argument version which uses only C_M.

- $F_S : \{0,1\}^n \to \{0,1\}^{l_S}$ is a random function
- $\pi : \{0,1\}^n \to \{0,1\}^n$ is a random permutation as before
- $\mathcal{E}_T(P_L, P_M) = (P_L \leftarrow\!\oplus F_S(\pi(P_M)), \pi(P_M))$ ie $\mathcal{E}_T = \theta_{\pi, F_S}$
- $\mathcal{D}_T(C_L, C_M) = (C_L \leftarrow\!\oplus F_S(C_M), \pi^{-1}(C_M))$ ie $\mathcal{D}_T = \theta^{-1}_{\pi, F_S}$

Since π is a permutation, for any pair of queries $C_M = C'_M$ if and only if $P_M = P'_M$. This is therefore indistinguishable from 5.1-rpswitch. This is a small upside to HPolyC's asymmetry: if a symmetrical construction such as $F_S(P_M \oplus \pi(P_M))$ were used here instead, it would be distinguishable with advantage $2^{-n}\binom{q}{2}$.

Summing these distances, with these constraints the advantage distinguishing between 5.1-randinner and 5.1-halfrf is at most $2^{-n}\binom{q}{2}$.

5.2 Collision finding

Theorem 3.3 of [Ber05b] shows that H is ϵ-almost-Δ-universal: for any $g \in \{0,1\}^n$ and any two distinct message m, m' such that $|m|, |m'| \leq kn$, there are at most $8k$ keys $K_H \in \mathcal{K}_H$ such that $H_{K_H}(m) \boxminus H_{K_H}(m') = g$. Since $|\mathcal{K}_H| = 2^{106}$, for a random $K_H \xleftarrow{\$} \mathcal{K}_H$ we have for any $(g, m) \neq (g', m')$ that $\Pr[H_{K_H}(m) \boxplus g = H_{K_H}(m') \boxplus g'] \leq 2^{-103}k$. We use this to bound the probability that the attacker will cause an "inner collision"—a query to the inner part which doesn't meet the constraints described in subsection 5.1.

From here on, we forbid only "pointless queries": after either of the queries $(C_L, C_R) \leftarrow \mathcal{E}_T(P_L, P_R)$ or $(P_L, P_R) \leftarrow \mathcal{D}_T(C_L, C_R)$, both the subsequent queries $\mathcal{E}_T(P_L, P_R)$ and $\mathcal{D}_T(C_L, C_R)$ would be forbidden. Again, we consider a computationally unbounded attacker.

A hash key $K_H \xleftarrow{\$} \mathcal{K}_H$ is chosen at random, and for each query we define

- $P_M = P_R \boxplus H_{K_H}(\text{encode}(T, P_L))$
- $C_M = C_R \boxplus H_{K_H}(\text{encode}(T, C_L))$

For query $1 \leq i \leq q$, we'll use superscripts to refer to the variables for that query, eg P_M^i, C_M^i. The attacker wins if there exists $i < j \leq q$ such that either

- j is a plaintext query (a query to $\mathcal{E}$) and $P_M^i = P_M^j$ or
- j is a ciphertext query (a query to $\mathcal{D}$) and $C_M^i = C_M^j$.

We do not consider a query such that $C_M^i = P_M^j$ (or vice versa) a win.

Let k be the maximum number of blocks processed by the hash, ie the least integer such that $|\text{encode}(T^i, P_L^i)| = |\text{encode}(T^i, C_L^i)| \leq kn$ for all i.

Here we are considering not distingushing probability but success probability; we show a bound on success probability for the first experiment, and for each subsequent experiment we bound the increase in success probability over the previous experiment.

5.2-keyend: Choose responses fairly at random of the appropriate length; once all q queries are complete, choose the hash key and see if the attacker succeeded.

If query j is a plaintext query, the attacker knows the query and result for all $i < j$, and can choose plaintext values to maximize the probability of success. If

$T^i, P_L^i = T^j, P_L^j$ then the hashes will be the same, and since pointless queries are forbidden we have that $P_R^i \neq P_R^j$ and therefore that $P_M^i \neq P_M^j$. Otherwise by the ϵAΔU property, $\Pr\left[P_M^i = P_M^j\right] \leq 2^{-103}k$. The same success bound holds for a ciphertext query. The overall probability of success is at most the sum of the probability of success for each pair: $2^{-103}k\binom{q}{2}$.

Note that we don't assume that probabilities are independent here; for any events A, B we have that $\Pr[A \vee B] \leq \Pr[A] + \Pr[B]$, with equality if A, B are disjoint. Here distjointness means that the attacker can choose queries such that no key causes more than one pair to succeed, making full use of the assumption on each query that previous queries have failed.

5.2-keystart: As with 5.2-keyend, but choose the key at the start of the experiment. This does not change the probability of success.

5.2-earlystop: As with 5.2-keystart, but end the experiment as soon as the attacker succeeds. This doesn't change the success probability.

5.2-randomfuncs: Use random function families for $\mathcal{E}$, $\mathcal{D}$. Since we forbid pointless queries, all responses of the appropriate length are equally likely as before, and this doesn't change the success probability.

5.2-randinner: Use random function families for $\mathcal{E}'$, $\mathcal{D}'$, and define

- $\mathcal{E}_T = \phi_{K_H,T}^{-1} \circ \mathcal{E}'_T \circ \phi_{K_H,T}$
- $\mathcal{D}_T = \phi_{K_H,T}^{-1} \circ \mathcal{D}'_T \circ \phi_{K_H,T}$

Since a random function composed with a bijective function is a random function, this doesn't change the success probability, which remains at most $2^{-103}k\binom{q}{2}$.

5.2-halfrf: The middle part of the sandwich is now a pair of random functions, as per the first experiment in subsection 5.1, 5.1-randinner. Substitute this with the last experiment, 5.1-halfrf:

- $F_S : \{0,1\}^n \to \{0,1\}^{l_S}$ is a random function
- $\pi : \{0,1\}^n \to \{0,1\}^n$ is a random permutation
- $\mathcal{E}_T = \phi_{K_H,T}^{-1} \circ \theta_{\pi,F_S} \circ \phi_{K_H,T}$
- $\mathcal{D}_T = \mathcal{E}_T^{-1} = \phi_{K_H,T}^{-1} \circ \theta_{\pi,F_S}^{-1} \circ \phi_{K_H,T}$

Given any attacker for this experiment, we construct a distinguisher between 5.1-randinner and 5.1-halfrf as follows: we choose a random K_H, and use our oracle for the inner part as θ. We report 1 if the attacker succeeds in generating an inner collision. We stop the experiment early if the attacker succeeds, and any query in which the attacker does not succeed is one that obeys the query bounds of subsection 5.1. Therefore, the difference in success probability between 5.2-randinner and 5.2-halfrf for these two outer experiments can be no more than the advantage bound of $2^{-n}\binom{q}{2}$ established in subsection 5.1 for distinguishing between 5.1-randinner and 5.1-halfrf. The success probability for this final experiment is therefore at most $(2^{-103}k + 2^{-n})\binom{q}{2}$.

5.3 Composition

Finally we put these pieces together to bound the advantage of distinguishing HPolyC from a family of random permutations. As before, the attacker can make encryption and decryption queries but "pointless" queries are forbidden; in addition, the attacker is constrained by a time bound t. We start with this experiment:

5.3-permutation: For all lengths and all T, $\mathcal{E}_T$ is a random permutation, and $\mathcal{D}_T = \mathcal{E}_T^{-1}$. The security of a variable-length tweakable SPRP is defined by the advantage bound in distinguishing it from this experiment.

5.3-randomfuncs: $\mathcal{E}$ and $\mathcal{D}$ are families of random functions. Since pointless queries are forbidden, the advantage in distinguishing this from 5.3-permutation is at most $2^{-|P|}\binom{q}{2} \leq 2^{-n}\binom{q}{2}$ per section C of [HR03].

5.3-randinner: $\mathcal{E}'$ and $\mathcal{D}'$ are families of random functions. Choose a hash key $K_H \xleftarrow{\$} \mathcal{K}_H$, and conjugate the random functions by Feistel calls to the hash function:

- $\mathcal{E}_T = \phi_{K_H,T}^{-1} \circ \mathcal{E}'_T \circ \phi_{K_H,T}$
- $\mathcal{D}_T = \phi_{K_H,T}^{-1} \circ \mathcal{D}'_T \circ \phi_{K_H,T}$

as per the step from 5.2-randomfuncs to 5.2-randinner. Since a random function composed with a bijective function is a random function, this is indistinguishable from 5.3-randomfuncs.

5.3-halfrf: Substitute 5.1-halfrf for 5.1-randinner.

- $K_H \xleftarrow{\$} \mathcal{K}_H$
- $F_S : \{0,1\}^n \to \{0,1\}^{l_S}$ is a random function
- $\pi : \{0,1\}^n \to \{0,1\}^n$ is a random permutation
- $\mathcal{E}_T = \phi_{K_H,T}^{-1} \circ \theta_{\pi,F_S} \circ \phi_{K_H,T}$
- $\mathcal{D}_T = \mathcal{E}_T^{-1}$

By subsection 5.2 the attacker's probability of causing an inner collision is at most $(2^{-103}k + 2^{-n})\binom{q}{2}$. If they do not cause a collision, their queries to θ meet the constraints set out in subsection 5.1, which bounds the distinguishing advantage in this case to $2^{-n}\binom{q}{2}$. The advantage in distinguishing from 5.3-randinner is at most the sum of the collision probability and the no-collision advantage, which is $(2^{-103}k + 2(2^{-n}))\binom{q}{2}$.

5.3-block: Choose a random $K_E \xleftarrow{\$} \mathcal{K}_E$ and substitute a block cipher E_{K_E} for π.

- $K_H \xleftarrow{\$} \mathcal{K}_H$, $K_E \xleftarrow{\$} \mathcal{K}_E$
- $F_S : \{0,1\}^n \to \{0,1\}^{l_S}$ is a random function
- $\mathcal{E}_T = \phi^{-1}_{K_H,T} \circ \theta_{E_{K_E},F_S} \circ \phi_{K_H,T}$
- $\mathcal{D}_T = \mathcal{E}_T^{-1}$

By the standard argument for such a substitution, the advantage for this substitution is at most $\mathsf{Adv}^{\pm\mathrm{prp}}_{E_{K_E}}(q, t')$ where t is a time bound on the attacker and $t' = t + \mathcal{O}\left(\sum_i |P^i| + |T^i|\right)$.

5.3-xrf: Use an XChaCha12-like random function to define the stream cipher and derive the keys K_H, K_E using the algorithm in subsection 2.5:

- $F_X : \{0,1\}^{192} \to \{0,1\}^{l_S}$ is a random function
- $\overline{K_H}||K_E||\ldots = F_X(1||0^{191})$
- $K_H = \mathrm{Poly1305Clamp}(\overline{K_H})$
- $F_S(C_M) = F_X(C_M||1||0^{63})$
- $\mathcal{E}_T = \phi^{-1}_{K_H,T} \circ \theta_{E_{K_E},F_S} \circ \phi_{K_H,T}$
- $\mathcal{D}_T = \mathcal{E}_T^{-1}$

This is indistinguishable from 5.3-block.

5.3-xchacha: Choose a random $K_S \xleftarrow{\$} \mathcal{K}_S$ and substitute $\mathrm{XChaCha12}_{K_S}$ for F_X.

- $K_S \xleftarrow{\$} \mathcal{K}_S$
- $\overline{K_H}||K_E||\ldots = \mathrm{XChaCha12}_{K_S}(1||0^{191})$
- $K_H = \mathrm{Poly1305Clamp}(\overline{K_H})$
- $F_S(C_M) = \mathrm{XChaCha12}_{K_S}(C_M||1||0^{63}) = S_{K_S}(C_M)$
- $\mathcal{E}_T = \phi^{-1}_{K_H,T} \circ \theta_{E_{K_E},S_{K_S}} \circ \phi_{K_H,T}$
- $\mathcal{D}_T = \mathcal{E}_T^{-1}$

This is HPolyC. Taking into account the 192-bit query to XChaCha12 to derive K_H, K_E, by the standard argument the advantage for this substitution is at most $\mathsf{Adv}^{\mathrm{prf}}_{\mathrm{XChaCha12}_{K_S}}(192 + \sum_i |P^i| - n, t')$ (where the first argument measures not queries but bits of output).

Summing these, the advantage in distinguishing HPolyC from a family of random permutations is at most $2^{-103}k\binom{q}{2} + 3(2^{-n})\binom{q}{2} + \mathsf{Adv}^{\pm\mathrm{prp}}_{E_{K_E}}(q, t') + \mathsf{Adv}^{\mathrm{prf}}_{\mathrm{XChaCha12}_{K_S}}(192 + \sum_i |P^i| - n, t')$. If the block and stream ciphers are secure, this will be dominated by the first term. Since $|P_L| + n = |P|$ but encode adds up to 159 bits, and padding for the last block of P_L another 127, $kn \leq \max_i(T^i + P^i + 158)$ and this first term is less than $2^{-111}q^2 \max_i(T^i + P^i + 158)$.

References

[AB96] Ross Anderson and Eli Biham. "Two practical and provably secure block ciphers: BEAR and LION". In: *Fast Software Encryption: Third International Workshop Cambridge, UK, February 21–23 1996 Proceedings*. Ed. by Dieter Gollmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 113–120. ISBN: 978-3-540-49652-6. DOI: 10.1007/3-540-60865-6_48. URL: https://www.cl.cam.ac.uk/~rja14/Papers/bear-lion.pdf.

[Bea+13] Ray Beaulieu et al. *The SIMON and SPECK Families of Lightweight Block Ciphers*. Tech. rep. 2013. URL: https://eprint.iacr.org/2013/404.

[Bea+15] Ray Beaulieu et al. *SIMON and SPECK: Block Ciphers for the Internet of Things*. Tech. rep. 2015. URL: https://eprint.iacr.org/2015/585.

[Bea+17] Ray Beaulieu et al. *Notes on the design and analysis of SIMON and SPECK*. Tech. rep. 2017. URL: https://eprint.iacr.org/2017/560.

[Bel+97] Mihir Bellare et al. "A Concrete Security Treatment of Symmetric Encryption". In: *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*. FOCS '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 394–. ISBN: 0-8186-8197-7. DOI: 10.1109/SFCS.1997.646128. URL: https://cseweb.ucsd.edu/~mihir/papers/sym-enc.html.

[Ber05a] Daniel J. Bernstein. *poly1305aes_test_clamp.c*. 2005. URL: https://cr.yp.to/mac/poly1305aes_test_clamp.c (visited on 05/31/2018).

[Ber05b] Daniel J. Bernstein. "The Poly1305-AES message-authentication code". In: *Fast software encryption: 12th international workshop, FSE 2005, Paris, France, February 21–23, 2005, revised selected papers*. Ed. by Henri Gilbert and Helena Handschuh. Vol. 3557. Lecture Notes in Computer Science. Springer, 2005, pp. 32–49. ISBN: 3–540–26541–4. URL: https://cr.yp.to/papers.html#poly1305.

[Ber06] Daniel J. Bernstein. *Salsa20/8 and Salsa20/12*. 2006. URL: https://cr.yp.to/snuffle/812.pdf (visited on 05/21/2018).

[Ber08a] Daniel J. Bernstein. "ChaCha, a variant of Salsa20". In: *State of the Art of Stream Ciphers Workshop, SASC 2008, Lausanne, Switzerland*. Jan. 2008. URL: https://cr.yp.to/papers.html#chacha.

[Ber08b] Daniel J. Bernstein. "The Salsa20 Family of Stream Ciphers". In: *New Stream Cipher Designs: The eSTREAM Finalists*. Ed. by Matthew Robshaw and Olivier Billet. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 84–97. ISBN: 978-3-540-68351-3. DOI:

10.1007/978-3-540-68351-3_8. URL: https://cr.yp.to/papers.html#salsafamily.

[Ber11] Daniel J. Bernstein. "Extending the Salsa20 nonce". In: *Workshop Record of Symmetric Key Encryption Workshop 2011*. 2011. URL: https://cr.yp.to/papers.html#xsalsa.

[BR99] Mihir Bellare and Phillip Rogaway. "On the Construction of Variable-Input-Length Ciphers". In: *Fast Software Encryption: 6th International Workshop, FSE'99 Rome, Italy, March 24–26, 1999 Proceedings*. Ed. by Lars Knudsen. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 231–244. ISBN: 978-3-540-48519-3. DOI: 10.1007/3-540-48519-8_17. URL: https://cseweb.ucsd.edu/~mihir/papers/lpe.pdf.

[BRS03] John Black, Phillip Rogaway, and Thomas Shrimpton. "Encryption-Scheme Security in the Presence of Key-Dependent Messages". In: *Selected Areas in Cryptography*. Ed. by Kaisa Nyberg and Howard Heys. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 62–75. ISBN: 978-3-540-36492-4. DOI: 10.1007/3-540-36492-7_6. URL: https://cise.ufl.edu/~teshrim/kdm.pdf.

[Cha+17] Debrup Chakraborty et al. *FAST: Disk Encryption and Beyond*. Tech. rep. 2017. URL: https://eprint.iacr.org/2017/849.

[CMS13] Debrup Chakraborty, Cuauhtemoc Mancillas-López, and Palash Sarkar. "STES: A Stream Cipher Based Low Cost Scheme for Securing Stored Data". In: *IEEE Transactions on Computers* 64 (2013), pp. 2691–2707. DOI: 10.1109/TC.2014.2366739.

[CN08] Debrup Chakraborty and Mridul Nandi. "An Improved Security Bound for HCTR". In: *Fast Software Encryption*. Ed. by Kaisa Nyberg. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 289–302. ISBN: 978-3-540-71039-4. DOI: 10.1007/978-3-540-71039-4_18. URL: https://www.iacr.org/cryptodb/archive/2008/FSE/paper/15611.pdf.

[Cro01] Paul Crowley. "Mercy: A Fast Large Block Cipher for Disk Sector Encryption". In: *Fast Software Encryption: 7th International Workshop, FSE 2000 New York, NY, USA, April 10–12, 2000 Proceedings*. Ed. by Gerhard Goos et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 49–63. ISBN: 978-3-540-44706-1. DOI: 10.1007/3-540-44706-7_4. URL: http://www.ciphergoth.org/crypto/mercy/.

[CS06] Debrup Chakraborty and Palash Sarkar. "A New Mode of Encryption Providing a Tweakable Strong Pseudo-random Permutation". In: *Fast Software Encryption: 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*. Ed. by Matthew Robshaw. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 293–309. ISBN: 978-3-540-36598-3. DOI: 10.1007/11799313_19. URL: https://eprint.iacr.org/2006/275.

[CS08] D. Chakraborty and P. Sarkar. "HCH: A New Tweakable Enciphering Scheme Using the Hash-Counter-Hash Approach". In: *IEEE Transactions on Information Theory* 54.4 (Apr. 2008), pp. 1683–1699. ISSN: 0018-9448. DOI: 10.1109/TIT.2008.917623. URL: https://eprint.iacr.org/2007/028.

[Dae+00] Joan Daemen et al. *Nessie Proposal: the block cipher* NOEKEON. Tech. rep. 2000. URL: http://gro.noekeon.org/.

[Den17] Frank Denis. *XChaCha20*. libsodium. 2017. URL: https://download.libsodium.org/doc/advanced/xchacha20.html (visited on 12/25/2017).

[Flu02] Scott R. Fluhrer. "Cryptanalysis of the Mercy Block Cipher". In: *Proc. Fast Software Encryption 2001, LNCS 2355*. Springer-Verlag, 2002, pp. 28–36. URL: https://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.6494.

[Hal05] Shai Halevi. "EME*: Extending EME to Handle Arbitrary-Length Messages with Associated Data". In: *Progress in Cryptology - INDOCRYPT 2004: 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004. Proceedings*. Ed. by Anne Canteaut and Kapaleeswaran Viswanathan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 315–327. ISBN: 978-3-540-30556-9. DOI: 10.1007/978-3-540-30556-9_25. URL: https://eprint.iacr.org/2004/125.

[Hal07] Shai Halevi. "Invertible Universal Hashing and the TET Encryption Mode". In: *Advances in Cryptology - CRYPTO 2007: 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007. Proceedings*. Ed. by Alfred Menezes. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 412–429. ISBN: 978-3-540-74143-5. DOI: 10.1007/978-3-540-74143-5_23. URL: https://eprint.iacr.org/2007/014.

[HR03] Shai Halevi and Phillip Rogaway. "A Tweakable Enciphering Mode". In: *Advances in Cryptology - CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings*. Ed. by Dan Boneh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 482–499. ISBN: 978-3-540-45146-4. DOI: 10.1007/978-3-540-45146-4_28. URL: https://eprint.iacr.org/2003/148.

[HR04] Shai Halevi and Phillip Rogaway. "A Parallelizable Enciphering Mode". In: *Topics in Cryptology – CT-RSA 2004: The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23-27, 2004, Proceedings*. Ed. by Tatsuaki Okamoto. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 292–304. ISBN: 978-3-540-24660-2. DOI: 10.1007/978-3-540-24660-2_23. URL: https://eprint.iacr.org/2003/147.

[IEE08] IEEE. *ANSI/IEEE 1619-2007 - IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices*. Tech. rep. 2008.

URL: https://standards.ieee.org/findstds/standard/1619-2007.html.

[LR88] Michael Luby and Charles Rackoff. "How to Construct Pseudorandom Permutations from Pseudorandom Functions". In: *SIAM J. Comput.* 17.2 (Apr. 1988), pp. 373–386. ISSN: 0097-5397. DOI: 10.1137/0217022. URL: https://github.com/emintham/Papers/blob/master/Luby%2CRackoff-%20How%20to%20Construct%20Pseudorandom%20Permutations%20from%20Pseudorandom%20Functions.pdf.

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. "Tweakable Block Ciphers". In: *Advances in Cryptology — CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings*. Ed. by Moti Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 31–46. ISBN: 978-3-540-45708-4. DOI: 10.1007/3-540-45708-9_3. URL: https://people.csail.mit.edu/rivest/pubs/LRW02.pdf.

[Luc96a] Stefan Lucks. "BEAST: A fast block cipher for arbitrary blocksizes". In: *Communications and Multimedia Security II: Proceedings of the IFIP TC6/TC11 International Conference on Communications and Multimedia Security at Essen, Germany, 23rd – 24th September 1996*. Ed. by Patrick Horster. Boston, MA: Springer US, 1996, pp. 144–153. ISBN: 978-0-387-35083-7. DOI: 10.1007/978-0-387-35083-7_13. URL: https://pdfs.semanticscholar.org/18fd/ac6eddb22687450c22e1135dc2d9c38c40d1.pdf.

[Luc96b] Stefan Lucks. "Faster Luby-Rackoff ciphers". In: *Fast Software Encryption*. Ed. by Dieter Gollmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 189–203. ISBN: 978-3-540-49652-6. DOI: 10.1007/3-540-60865-6_53. URL: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.7485.

[LWR00] Helger Lipmaa, David Wagner, and Phillip Rogaway. *Comments to NIST concerning AES modes of operation: CTR-mode encryption*. 2000. URL: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.79.1353.

[Mau93] Ueli M. Maurer. "A Simplified and Generalized Treatment of Luby-Rackoff Pseudorandom Permutation Generators". In: *Proceedings of the 11th Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT'92. Balatonfüred, Hungary: Springer-Verlag, 1993, pp. 239–255. ISBN: 3-540-56413-6. URL: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.6117.

[MF07] David A. McGrew and Scott R. Fluhrer. "The Security of the Extended Codebook (XCB) Mode of Operation". In: *Selected Areas in Cryptography: 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers*. Ed. by Carlisle Adams, Ali Miri, and Michael Wiener. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 311–327. ISBN: 978-3-540-77360-3. DOI:

10.1007/978-3-540-77360-3_20. URL:
https://eprint.iacr.org/2007/298.

[Nan08] Mridul Nandi. *Improving upon HCTR and matching attacks for Hash-Counter-Hash approach*. Tech. rep. 2008. URL: https://eprint.iacr.org/2008/090.

[NIS01] NIST. *Advanced Encryption Standard (AES)*. National Institute of Standards and Technology. FIPS Publication 197, Nov. 2001. URL: https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf.

[NL15] Y. Nir and A. Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 7539. RFC Editor, May 2015. URL: https://www.rfc-editor.org/rfc/rfc7539.txt.

[NR99] Moni Naor and Omer Reingold. "On the Construction of Pseudorandom Permutations: Luby–Rackoff Revisited". In: *Journal of Cryptology* 12.1 (Jan. 1999), pp. 29–66. ISSN: 1432-1378. DOI: 10.1007/PL00003817. URL: https://omereingold.files.wordpress.com/2014/10/lr.pdf.

[NW97] Roger M. Needham and David J. Wheeler. *Tea extensions*. Tech. rep. 1997. URL: http://www.cix.co.uk/~klockstone/xtea.pdf.

[Pat91] Jacques Patarin. "Pseudorandom permutations based on the D.E.S. scheme". In: *EUROCODE '90*. Ed. by Gérard Cohen and Pascale Charpin. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 193–204. ISBN: 978-3-540-47546-0. DOI: 10.1007/3-540-54303-1_131.

[Sar07] Palash Sarkar. "Improving Upon the TET Mode of Operation". In: *Information Security and Cryptology - ICISC 2007: 10th International Conference, Seoul, Korea, November 29-30, 2007. Proceedings*. Ed. by Kil-Hyun Nam and Gwangsoo Rhee. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 180–192. ISBN: 978-3-540-76788-6. DOI: 10.1007/978-3-540-76788-6_15. URL: https://eprint.iacr.org/2007/317.

[Sar09] Palash Sarkar. *Tweakable Enciphering Schemes From Stream Ciphers With IV*. Tech. rep. 2009. URL: https://eprint.iacr.org/2009/321.

[Sar11] Palash Sarkar. "Tweakable enciphering schemes using only the encryption function of a block cipher". In: *Information Processing Letters* 111.19 (2011), pp. 945–955. ISSN: 0020-0190. DOI: https://doi.org/10.1016/j.ipl.2011.06.014.

[Sch98] Rich Schroeppel. *Hasty Pudding Cipher Specification*. 1998. URL: http://richard.schroeppel.name/hpc/hpc-spec (visited on 05/21/2018).

[Sho04] Victor Shoup. *Sequences of games: a tool for taming complexity in security proofs*. Tech. rep. 2004. URL: https://eprint.iacr.org/2004/332.

[WFW05] Peng Wang, Dengguo Feng, and Wenling Wu. "HCTR: A Variable-Input-Length Enciphering Mode". In: *Information Security* *and Cryptology: First SKLOIS Conference, CISC 2005, Beijing, China, December 15-17, 2005. Proceedings*. Ed. by Dengguo Feng, Dongdai Lin, and Moti Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 175–188. ISBN: 978-3-540-32424-9. DOI: 10.1007/11599548_15. URL: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.470.5288.

Version: 90a461bf57f28bf222b03b73fa5d66dacc45db5a