Adiantum: length-preserving encryption for entry-level processors

Paul Crowley and Eric Biggers

Google LLC

November 10, 2018

Abstract

We present HBSH, a simple construction for tweakable length-preserving encryption which directly supports the fastest options for hashing and stream encryption for processors without AES or other crypto instructions, with a provable quadratic advantage bound. Our composition Adiantum uses NH, Poly1305, XChaCha12, and a single AES invocation. On an ARM Cortex-A7 processor, Adiantum decrypts 4096-byte messages at 10.6 cycles per byte, over five times faster than AES-256-XTS, with a constant-time implementation. We also define HPolyC which is simpler and has excellent key agility at 13.6 cycles per byte.

This paper: `https://ia.cr/2018/720`
Source: `https://github.com/google/adiantum`

Email: {paulcrowley,ebiggers}@google.com

Contents

1 Introduction **2**
- 1.1 History . . . 2
- 1.2 Our contribution . . . 3
- 1.3 Implementation and test vectors . . . 4

2 Specification **4**

3 Design **6**

4 Performance **7**

5 Security reduction **9**
- 5.1 Definitions . . . 9
- 5.2 Primary claim . . . 11
- 5.3 H-coefficient technique . . . 11
- 5.4 Preliminaries . . . 12
- 5.5 Lemmas . . . 13

5.6 Proof of primary claim 16

References **16**

A ϵAΔU functions for HBSH **22**

A.1 Notation 22
A.2 Poly1305 22
A.3 HPolyC hashing 23
A.4 NH 23
A.5 Adiantum hashing 24

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, hash functions such as NH [Kro00] and 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 the HBSH (hash, block cipher, stream cipher, hash) construction, which generalizes over constructions such as HCTR and HCH by taking an ϵ-almost-Δ-universal hash function and a nonce-accepting stream cipher as components. Based on this construction, our main proposal is Adiantum, which uses a combination of NH and Poly1305 for the hashing, XChaCha12 for the stream cipher, and AES for the single block cipher application. Adiantum:

- 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 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, Adiantum handles:

- any message and tweak lengths within the allowed range,
- varying message and tweak lengths for the same keys.

We also describe a simpler proposal, HPolyC, which sacrifices a little speed on large blocks for simplicity and greater key agility, leaving out the NH hash layer.

1.3 Implementation and test vectors

Implementations in Python, C, and ARMv7 assembly, as well as thousands of test vectors and the LaTeX source for this paper, are available from our source code repository at `https://github.com/google/adiantum`.

2 Specification

The HBSH construction is shown in Figure 1 and Figure 2. From plaintext P of at least n bits and a tweak T, it generates a ciphertext C of the same length as P. HBSH divides the plaintext into a right-hand block of n bits and a left-hand block with the remainder of the input, and applies an unbalanced Feistel network. It uses an ϵ-almost-Δ-universal function H, an n-bit block cipher E, and a stream cipher S. P_R, P_M, C_M, C_R are n bits long.

Notation: 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 $f_a^{-1}(f_a(b)) = b$. $||$ represents concatenation, and λ the empty string. $|X|$ represents the length of $X \in \{0,1\}^*$ in bits. $Y[a;l]$ refers to the subsequence of Y of length l starting at a. $X \hookleftarrow\!\!\oplus Y$ is $X \oplus Y[0;|X|]$. $\mathrm{pad}_l(X) = X||0^v$ where v is the least integer ≥ 0 such that l divides $|X| + v$.

Hash: $H : \mathcal{K}_H \times \mathcal{T} \times \mathcal{L} \to \{0,1\}^n$ is an ϵ-almost-Δ-universal (ϵAΔU) function (as defined in subsection 5.1) yielding a group element represented as an n-bit string. $\boxplus$ represents addition in a group which depends on the hash function, and $\boxminus$ subtraction.

HPolyC and Adiantum differ only in their choice of hash function. HPolyC is based on Poly1305, while Adiantum uses both Poly1305 and NH; specifically little-endian $\mathrm{NH}^T[256, 32, 4]$ with a stride of 2 for fast vectorization. In both cases, the group used for $\boxplus$ and $\boxminus$ is $\mathbb{Z}/2^{128}\mathbb{Z}$. The value of ϵ depends on bounds on the input lengths. We defer full details to Appendix A.

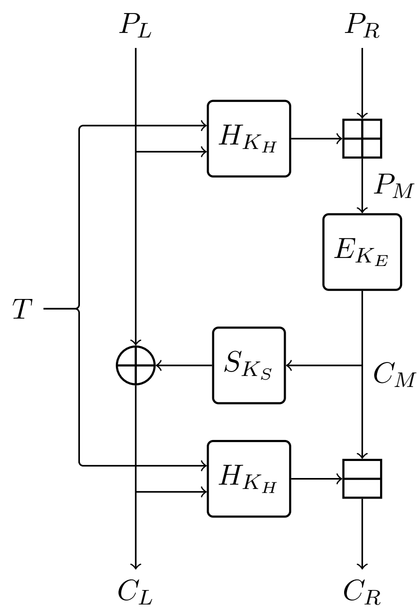

Figure 1: HBSH

procedure HBSHENCRYPT(T, P)
  $P_L || P_R \leftarrow P$
  $P_M \leftarrow P_R \boxplus H_{K_H}(T, P_L)$
  $C_M \leftarrow E_{K_E}(P_M)$
  $C_L \leftarrow P_L \stackrel{\leftarrow}{\oplus} S_{K_S}(C_M)$
  $C_R \leftarrow C_M \boxminus H_{K_H}(T, C_L)$
  $C \leftarrow C_L || C_R$
  return C
end procedure
procedure HBSHDECRYPT(T, C)
  $C_L || C_R \leftarrow C$
  $C_M \leftarrow C_R \boxplus H_{K_H}(T, C_L)$
  $P_L \leftarrow C_L \stackrel{\leftarrow}{\oplus} S_{K_S}(C_M)$
  $P_M \leftarrow E^{-1}_{K_E}(C_M)$
  $P_R \leftarrow P_M \boxminus H_{K_H}(T, P_L)$
  $P \leftarrow P_L || P_R$
  return P
end procedure

Figure 2: Pseudocode for HBSH

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

Stream cipher and message space: $S : \mathcal{K}_S \times \mathcal{N} \to \{0,1\}^{l_S}$ is a stream cipher which takes a key and a nonce and produces a long random stream. In normal use the nonce is an n-bit string, but for key derivation we use the empty string λ, which is distinct from all n-bit strings; thus $\lambda \cup \{0,1\}^n \subseteq \mathcal{N}$. P_L, C_L must therefore be in the space $\mathcal{L} = \bigcup_{i=0}^{l_S} \{0,1\}^i$ and the space of HBSH plaintexts and ciphertexts $P, C \in \mathcal{M} = \bigcup_{i=0}^{l_S} \{0,1\}^{i+n}$.

Adiantum and HPolyC use the XChaCha12 stream cipher. The ChaCha [Ber08a] stream ciphers takes a 64-bit nonce, and RFC7539 [NL15] proposes a ChaCha20 variant with a 96-bit nonce, but we need a 128-bit nonce. The XSalsa20 construction [Ber11] proposed for Salsa20 [Ber08b; Ber06] extends the nonce to 192 bits, and applies straightforwardly to ChaCha [Arc18; Vai17; Den17]. We then construct a function that takes a variable-length nonce of up to 191 bits by padding with a 1 followed by zeroes: $S_{K_S}(C_M) = \mathrm{XChaCha12}_{K_S}(\mathrm{pad}_{192}(C_M||1))$. For a given key and nonce, XChaCha12 produces $l_S = 2^{73}$ bits of output.

Key derivation: HBSH derives K_E and K_H from K_S using a zero-length nonce: $K_E||K_H||\ldots = S_{K_S}(\lambda)$. An earlier version of this paper used $K_H||K_E||\ldots = S_{K_S}(\lambda)$ for HPolyC.

3 Design

Three-pass structure: 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.

Block cipher: [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. If the plaintext is exactly n bits long, the stream cipher is not used, and the construction becomes $C = E_{K_E}(P \boxplus H_{K_H}(T, \lambda)) \boxminus H_{K_H}(T, \lambda)$. 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.

Building blocks: It's unusual for a construction to require more than two distinct building blocks. More commonly, a hash-XOR-hash mode uses the block cipher to build a stream cipher (eg using CTR mode [LWR00]) as well as using it directly on the narrow side of the block. 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.

KDM security: We do not consider an attack model in which derived keys are presented as input. 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.

Hash function: Since the ϵAΔU is run twice over the bulk of the block, its speed is especially crucial for large blocks. One of the fastest such functions in software is NH, and it's also appealingly simple; however as discussed in subsection A.4 it generally has to be combined with a second hashing stage, and for this purpose we use Poly1305. The 1KiB block size used for NH means that we can use a simple, portable implementation of Poly1305 without a great cost in speed. We considered using UHASH (as defined for UMAC [Kro06]) rather than our custom combination of NH and Poly1305; however, available UHASH implementations are not constant-time, and a constant-time implementation would be significantly slower.

Key agility: For the 4KiB blocks of disk encryption, the 1KiB NH key size has only a small impact on key agility. Applications that need high key agility even on small blocks may instead use HPolyC, which uses Poly1305 directly. For this a vectorized Poly1305 implementation is important. The main cost of a new HPolyC key is a single XChaCha12 invocation to generate subkeys. 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 encryption or decryption. No extra work is required for differing message or tweak lengths for either Adiantum or HPolyC.

Constant-time: NH, Poly1305 and ChaCha12 are 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, Adiantum and HPolyC will inherit security against this class of side-channel attacks.

Parallelizability: NH, 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.

Naming: "Adiantum" is the genus of the maidenhair fern, which in the language of flowers (floriography) signifies sincerity and discretion. [Tou19]

4 Performance

In Table 1 we show performance on an ARM Cortex-A7 processor in the Snapdragon 400 chipset running at 1.19 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 devices, and is representative of the kind of platform we mean to target. Where the figures are within 2%, 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)
NH	1.3	1.4
Poly1305	2.9	3.3
ChaCha8	5.1	5.2
ChaCha12	7.1	7.2
Adiantum-XChaCha8-AES	8.5	13.2
Adiantum-XChaCha12-AES	**10.6**	**15.8**
ChaCha20	11.2	11.3
HPolyC-XChaCha8-AES	11.5	16.5
HPolyC-XChaCha12-AES	**13.6**	**18.7**
Adiantum-XChaCha20-AES	14.7	20.2
Speck128/128-XTS	15.0	16.1
Speck128/256-XTS	15.8	16.9
HPolyC-XChaCha20-AES	17.8	23.4
NOEKEON-XTS	26.9	27.9
XTEA-XTS	28.7	29.7
AES-128-XTS (encryption)	36.1	37.2
AES-128-XTS (decryption)	42.7	43.9
AES-256-XTS (encryption)	48.9	50.5
AES-256-XTS (decryption)	58.6	60.1

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, NH, 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. As an exception, given the high difficulty of writing truly constant-time AES software [Ber05a], for single-block AES we tolerate an implementation that merely prefetches the lookup tables as a hardening measure. In every case the performance-critical parts were written in assembly language, usually using NEON instructions. Our tests complete processing of each message before starting the next, so latency of a single message in cycles is the product of message size and cpb.

Adiantum and HPolyC are the only algorithms in Table 1 that are tweakable super-pseudorandom permutations over the entire sector. We expect any

Algorithm	Source	Notes
ChaCha	Linux v4.17	`chacha20-neon-core.S`, modified to support ChaCha8 and ChaCha12; also applied optimizations from `cryptodev` commit a1b22a5f45fe8841
Poly1305	OpenSSL 1.1.0h	`poly1305-armv4.S`, modified to precompute key powers just once per key
AES	Linux v4.17	`aes-cipher-core.S`, modified to prefetch lookup tables
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

AES-based construction to that end to be significantly slower than AES-XTS.

We conclude that for 4096-byte sectors, Adiantum (aka Adiantum-XChaCha12-AES) can perform significantly better than an aggressively designed block cipher (Speck128/256) in XTS mode. Efficient implementations of NH, 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. Similarly, one could substitute NOEKEON in place of AES-256 to make defense against timing attacks easier and improve performance. This may weaken security against a brute-force attack since NOEKEON has only a 128-bit key, though it's not obvious how to mount such an attack when the hashing and stream cipher keys are unknown. Note that this is a different axis of security than success probability; an attack that needs (say) 2^{40} work and always succeeds is a much bigger problem than than an attack that needs negligible work and succeeds with probability 2^{-40}.

5 Security reduction

5.1 Definitions

Below we define the security properties of HBSH and the primitives it uses, and prove a relationship the two. In what follows we draw on definitions used in [CN08].

Hash function: The hash function $H : \mathcal{K}_H \times \mathcal{T} \times \mathcal{L} \rightarrow \{0,1\}^n$ must be ϵ-almost-Δ-universal for some ϵ: for any $g \in \{0,1\}^n$ and any two distinct messages $(T, L) \neq (T', L')$:

$$\Pr_{K \leftarrow\!\$\, \mathcal{K}_H}\left[H_K(T, L) \boxminus H_K(T', L') = g\right] \leq \epsilon$$

In general the value of ϵ will depend on bounds on the lengths of T and L.

Stream cipher: The stream cipher $S : \mathcal{K}_S \times \mathcal{N} \rightarrow \{0,1\}^{l_S}$ must be a pseudorandom function.

$$\begin{aligned}\mathsf{Adv}^{\mathrm{prf}}_S(A) &\stackrel{\mathrm{def}}{=} \Big|\Pr_{K \leftarrow\!\$\, \mathcal{K}_S}\left[A^{S_K} \Rightarrow 1\right] \\ &\qquad - \Pr_{F \leftarrow\!\$\, (\mathcal{N} \rightarrow \{0,1\}^{l_S})}\left[A^F \Rightarrow 1\right]\Big| \\ \mathsf{Adv}^{\mathrm{prf}}_S(q, l, t) &\stackrel{\mathrm{def}}{=} \max_{A \in \mathcal{A}(q,l,t)} \mathsf{Adv}^{\mathrm{prf}}_S(A)\end{aligned}$$

where A is an adversary, $\mathcal{N} \rightarrow \{0,1\}^{l_S}$ denotes the set of all functions from $\mathcal{N}$ to $\{0,1\}^{l_S}$, and $\mathcal{A}(q, l, t)$ is the set of all adversaries that make at most q queries, discard all but l bits from the results of those queries, and take at most t time.

Block cipher: The block cipher $E : \mathcal{K}_E \times \{0,1\}^n \rightarrow \{0,1\}^n$ must be a super-pseudorandom permutation.

$$\begin{aligned}\mathsf{Adv}^{\pm\mathrm{prp}}_E(A) &\stackrel{\mathrm{def}}{=} \Big|\Pr_{K \leftarrow\!\$\, \mathcal{K}_E}\left[A^{E_K, E_K^{-1}} \Rightarrow 1\right] \\ &\qquad - \Pr_{\pi \leftarrow\!\$\, \mathrm{Perm}(\{0,1\}^n)}\left[A^{\pi, \pi^{-1}} \Rightarrow 1\right]\Big| \\ \mathsf{Adv}^{\pm\mathrm{prp}}_E(q, t) &\stackrel{\mathrm{def}}{=} \max_{A \in \mathcal{A}(q,t)} \mathsf{Adv}^{\pm\mathrm{prp}}_E(A)\end{aligned}$$

where A is an adversary, $\mathrm{Perm}(S)$ denotes the set of all permutations on a set S, and $\mathcal{A}(q, t)$ is the set of all adversaries that make at most q queries and take at most t time.

Tweakable SPRP: Let $\mathrm{LP}^{\mathcal{T}}(\mathcal{M})$ denote the set of all tweakable length-preserving functions $\boldsymbol{f} : \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{M}$ such that for all $T, M \in \mathcal{T} \times \mathcal{M}$, $|\boldsymbol{f}(T, M)| = |M|$. Let $\mathrm{Perm}^{\mathcal{T}}(\mathcal{M})$ denote the set of $\boldsymbol{\pi} \in \mathrm{LP}^{\mathcal{T}}(\mathcal{M})$ such that for all $T \in \mathcal{T}$, $\boldsymbol{\pi}_T$ is a bijection. In an abuse of notation we use $\boldsymbol{\pi}^{-1}$ to refer to the function such that $\boldsymbol{\pi}^{-1}(T, \boldsymbol{\pi}(T, M)) = M$ ie $(\boldsymbol{\pi}^{-1})_T = (\boldsymbol{\pi}_T)^{-1}$.

For a tweakable, variable-input-length, super-pseudorandom permutation $\boldsymbol{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{M}$ the distinguishing advantage of an adversary A is:

$$\begin{aligned}\mathsf{Adv}^{\pm\widetilde{\mathrm{prp}}}_{\boldsymbol{E}}(A) &\stackrel{\mathrm{def}}{=} \Big|\Pr_{K \leftarrow\!\$\, \mathcal{K}}\left[A^{\boldsymbol{E}_K, \boldsymbol{E}_K^{-1}} \Rightarrow 1\right] \\ &\qquad - \Pr_{\boldsymbol{\pi} \leftarrow\!\$\, \mathrm{Perm}^{\mathcal{T}}(\mathcal{M})}\left[A^{\boldsymbol{\pi}, \boldsymbol{\pi}^{-1}} \Rightarrow 1\right]\Big|\end{aligned}$$

and

$$\mathsf{Adv}^{\pm\widetilde{\mathrm{prp}}}_{\boldsymbol{E}}(q, l_T, l_M, t) \stackrel{\mathrm{def}}{=} \max_{A \in \mathcal{A}(q, l_T, l_M, t)} \mathsf{Adv}^{\pm\widetilde{\mathrm{prp}}}_{\boldsymbol{E}}(A)$$

where $\mathcal{A}(q, l_T, l_M, t)$ is the set of all adversaries that make at most q queries with tweak of length at most l_T and message of length at most l_M and take at most t time.

5.2 Primary claim

Theorem 1. *Where HBSH mode is instantiated with hash function H, block cipher E and stream cipher S, and where H is ϵ-almost-Δ-universal for inputs such that $|T| \leq l_T$, $|L| \leq l_M - n$, then:*

$$\begin{aligned}\mathsf{Adv}^{\pm\widetilde{\mathrm{prp}}}_{\mathrm{HBSH}}(q, l_T, l_M, t) \leq & (\epsilon + 2(2^{-n}))\binom{q}{2} \\ & + \mathsf{Adv}^{\mathrm{prf}}_{S}(q+1, |K_E| + |K_H| + q(l_M - n), t') \\ & + \mathsf{Adv}^{\pm\mathrm{prp}}_{E}(q, t')\end{aligned}$$

where $t' = t + \mathcal{O}(q(l_T + l_M))$.

Proof. Deferred to subsection 5.6.

5.3 H-coefficient technique

The H-coefficient technique was introduced by Patarin in 1991 [Pat91; Pat09]. In what follows we rely on the exposition of [CS14] Section 3, "The H-coefficient Technique in a Nutshell", though we vary slightly by introducing a new symbol Υ so we can distinguish between what is sampled and the adversary oracles.

WLOG we assume a deterministic adversary A. We wish to bound the adversary's ability to distinguish between two "worlds", world X (the "real world") and world Y (the "ideal world"). Associated with world X we have

- Ω_X: a set of instances we sample fairly from. We write $\Pr_{\Omega_X}$ as shorthand for $\Pr_{\omega \leftarrow\$ \Omega_X}$.
- Υ_X: a map from an instance $\omega \in \Omega_X$ to a tuple of deterministic oracles we can present to the adversary.
- $\rho_X \stackrel{\text{def}}{=} \Pr_{\Omega_X}\left[A^{\Upsilon_X(\omega)} \Rightarrow 1\right]$ where the adversary A is clear from context. As the adversary interacts with the oracles, a transcript τ of queries and responses is generated.
- X: a random variable representing a transcript for $A^{\Upsilon_X(\omega)}$ given $\omega \leftarrow\$ \Omega_X$; we write $\tau \sim X$ to indicate that τ is sampled from this distribution.

- comp_X: We write $\omega \in \mathsf{comp}_X(\tau)$ if a transcript τ is “compatible” with an instance $\omega \in \Omega_X$, ie if given an adversary A that makes those queries, the oracle $\Upsilon_X(\omega)$ makes those responses and thus $A^{\Upsilon_X(\omega)}$ produces that transcript.

We have the same for world Y throughout.

The H-coefficient technique is based on the observation that:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Y}}_{\mathsf{X}}(A) &= |\rho_X - \rho_Y| \\ &\leq \sum_{\tau:\Pr[Y=\tau]>\Pr[X=\tau]} (\Pr[Y=\tau] - \Pr[X=\tau]) \\ &= 1 - \mathbb{E}_{\tau\sim Y}\left[\min\left(1, \frac{\Pr_{\Omega_X}[\omega \in \mathsf{comp}_X(\tau)]}{\Pr_{\Omega_Y}[\omega \in \mathsf{comp}_Y(\tau)]}\right)\right]\end{aligned}$$

as shown in [CS14].

5.4 Preliminaries

We use this technique to prove a distinguishing bound between random query responses and an “idealized” version of HBSH that uses a random function and permutation in place of pseudorandom primitives.

Transcript: Our transcript τ is a sequence of tuples (d^i, T^i, P^i, C^i) in $\{+,-\} \times \mathcal{T} \times \mathcal{M} \times \mathcal{M}$ for $i \in [0 \ldots q-1]$. For the ith sequential query d^i is the direction of the query: if $d^i = +$ then a plaintext query T^i, P^i is made and the result is C^i, while if $d^i = -$ then a ciphertext query T^i, C^i is made and the result is P^i.

Pointless queries: We consider adversaries contained in $\mathcal{A}(q, l_T, l_M, t)$ for some value of the bounds q, l_T, l_M, t. Without loss of generality, we consider only adversaries who do not make “pointless” queries as defined in [HR03]. Thus for $i < j$, if $d^j = +$ then $(T^j, P^j) \neq (T^i, P^i)$, and similarly if $d^j = -$ then $(T^j, C^j) \neq (T^i, C^i)$.

Helper functions: We define here helper functions ξ, θ, ϕ, and η, useful for constructing HBSH-like ciphers. Where a parameter is given as $L||R$, $|R| = n$.

$$\begin{aligned}\xi :& \mathcal{K}_H \times \mathcal{T} \times \mathcal{M} \to \{0,1\}^n \\ \xi_{K_H}(T, L||R) \stackrel{\text{def}}{=}& R \boxplus H_{K_H}(T, L)\end{aligned}$$

$$
\begin{aligned}
\phi :& \mathcal{K}_H \times \mathcal{T} \times \mathcal{M} \to \mathcal{M} \\
\phi_{K_H,T}(L||R) \stackrel{\text{def}}{=}& L||\xi_{K_H}(T, L||R) \\
=& L||(R \boxplus H_{K_H}(T, L)) \\
\phi^{-1}_{K_H,T}(L||R) =& L||(R \boxminus H_{K_H}(T, L))
\end{aligned}
$$

$$
\begin{aligned}
\theta :& \operatorname{Perm}(\{0,1\}^n) \times (\mathcal{N} \to \{0,1\}^{l_S}) \times \mathcal{M} \to \mathcal{M} \\
\theta_{\pi,F}(L||R) \stackrel{\text{def}}{=}& (L \leftarrow\!\oplus F(\pi(R)))||\pi(R)
\end{aligned}
$$

$$
\begin{aligned}
\eta :& \mathcal{K}_H \times \operatorname{Perm}(\{0,1\}^n) \times (\mathcal{N} \to \{0,1\}^{l_S}) \times \mathcal{T} \times \mathcal{M} \to \mathcal{M} \\
\eta_{K_H,\pi,F,T} \stackrel{\text{def}}{=}& \phi^{-1}_{K_H,T} \circ \theta_{\pi,F} \circ \phi_{K_H,T}
\end{aligned}
$$

Bad events: We define two bad events **badQ** and **badR**.

- $(K_H, \tau) \in$ **badQ** if there exists $i < j$ such that
  - either $d^j = +$ and $\xi_{K_H}(T^i, P^i) = \xi_{K_H}(T^j, P^j)$
  - or $d^j = -$ and $\xi_{K_H}(T^i, C^i) = \xi_{K_H}(T^j, C^j)$.
- $(K_H, \tau) \in$ **badR** if there exists $i < j$ such that
  - either $d^j = +$ and $\xi_{K_H}(T^i, C^i) = \xi_{K_H}(T^j, C^j)$
  - or $d^j = -$ and $\xi_{K_H}(T^i, P^i) = \xi_{K_H}(T^j, P^j)$.

Finally we define the disjunction **bad** $\stackrel{\text{def}}{=}$ **badQ** $\cup$ **badR**.

Ideal world: Our "ideal world" samples fairly from all possible pairs of length-preserving functions: $\Omega_Y \stackrel{\text{def}}{=} \mathrm{LP}^{\mathcal{T}}(\mathcal{M}) \times \mathrm{LP}^{\mathcal{T}}(\mathcal{M})$, so given $(\mathcal{E}, \mathcal{D}) \in \Omega_Y$, Υ_Y is simply the identity function: $\Upsilon_Y(\mathcal{E}, \mathcal{D}) \stackrel{\text{def}}{=} \mathcal{E}, \mathcal{D}$

Real world: Our "real world" is an idealized form of HBSH which uses a random function and permutation: $\Omega_X \stackrel{\text{def}}{=} \mathcal{K}_H \times \operatorname{Perm}(\{0,1\}^n) \times (\mathcal{N} \to \{0,1\}^{l_S})$, and given $(K_H, \pi, F) \in \Omega_X$, $\Upsilon_X(K_H, \pi, F) \stackrel{\text{def}}{=} \eta_{K_H,\pi,F}, \eta^{-1}_{K_H,\pi,F}$

5.5 Lemmas

Lemma 1. *For any τ such that $\Pr[Y = \tau] > 0$,*

$$\Pr_{K_H \leftarrow\$ \mathcal{K}_H}[(K_H, \tau) \in \mathbf{badQ}] \le \epsilon \binom{q}{2}$$

Proof. Assume $d^j = +$ for some pair i, j, and let $L^i||R^i = P^i$ and similarly for P^j. From $\Pr[Y = \tau] > 0$ we know that $|T^i|, |T^j| \le l_T$ and $|P^i|, |P^j| \le l_M$, and therefore that $|L^i|, |L^j| \le l_M - n$. Because pointless queries are forbidden we also know that $(T^i, P^i) \neq (T^j, P^j)$.

$$\begin{aligned}
&\xi_{K_H}(T^i, L^i||R^i) = \xi_{K_H}(T^j, L^j||R^j) \\
\Leftrightarrow &R^i \boxplus H_{K_H}(T^i, L^i) = R^j \boxplus H_{K_H}(T^j, L^j) \\
\Leftrightarrow &H_{K_H}(T^i, L^i) \boxminus H_{K_H}(T^j, L^j) = R^j \boxminus R^i
\end{aligned}$$

If $(T^i, L^i) = (T^j, L^j)$ then $R^i \neq R^j$ and equality cannot occur. Otherwise by the ϵAΔU property of H this occurs with probability at most ϵ (where ϵ depends on the bounds on the parameters l_T, $l_M - n$).

Where $d^j = -$, a similar argument applies for C^i, C^j. For an upper bound, we sum across all $\binom{q}{2}$ pairs i, j. □

Lemma 2. *For any $K_H \leftarrow_\$ \mathcal{K}_H$,*

$$\Pr_{\tau \sim Y}[(K_H, \tau) \in \mathbf{badR}] \le 2^{-n}\binom{q}{2}$$

Proof. Assume $d^j = +$ for some pair i, j, and let $L^i||R^i = C^i$ and similarly for C^j. Because pointless queries are forbidden, in the ideal world, conditioning on all prior queries and responses, all possible values of C^j such that $|C^j| = |P^j|$ will be equally likely. In particular, even after conditioning on L^j, all values of R^j are equally likely. Therefore $\Pr\left[R^j = R^i \boxplus H_{K_H}(T^i, L^i) \boxminus H_{K_H}(T^j, L^j)\right] = 2^{-n}$.

Where $d^j = -$, a similar argument applies for P^i, P^j. For an upper bound, we sum across all $\binom{q}{2}$ pairs i, j. □

Lemma 3. *For any $K_H \in \mathcal{K}_H$ and transcript τ such that $\Pr[Y = \tau] > 0$ and $(K_H, \tau) \notin \mathbf{bad}$,*

$$\Pr_{\Omega_X}[\omega \in \mathsf{comp}_X(\tau) \mid \omega = (K_H, ., .)] \ge \Pr_{\Omega_Y}[\omega \in \mathsf{comp}_Y(\tau)]$$

Proof. In the ideal world, for any transcript such that $\Pr[Y = \tau] > 0$, since all queries are distinct, the responses are independent of all previous responses, and $\Pr_{\Omega_Y}[\omega \in \mathsf{comp}_Y(\tau)] = \prod_i 2^{-|P^i|}$. Let $P_L^i||P_R^i = P^i$, $P_M^i = \xi_{K_H,T^i}(P^i)$ and similarly for C^i. Since $(K_H, \tau) \notin \mathbf{bad}$ we have that $P_M^i \neq P_M^j$ and $C_M^i \neq C_M^j$ for all $i \neq j$.

$$\begin{aligned}
&\eta_{K_H,\pi,F,T^i}(P^i) = C^i \\
\Leftrightarrow &\phi^{-1}_{K_H,T^i}(\theta_{\pi,F}(\phi_{K_H,T^i}(P^i))) = C^i \\
\Leftrightarrow &\theta_{\pi,F}(P_L^i||P_M^i) = C_L^i||C_M^i \\
\Leftrightarrow &\pi(P_M^i) = C_M^i \wedge F(C_M^i)[0; |P^i| - n] = P_L^i \oplus C_L^i
\end{aligned}$$

These conditions are independent, since they depend on independently drawn variables:

$$\Pr_{F \leftarrow\$ (\mathcal{N} \to \{0,1\}^{l_S})}\left[\forall_i : F(C_M^i)[0; |P^i| - n] = P_L^i \oplus C_L^i\right] = \prod_i 2^{-(|P^i|-n)}$$

and

$$\Pr_{\pi \leftarrow\$ \text{ Perm}(\{0,1\}^n)}\left[\forall_i : \pi(P_M^i) = C_M^i\right] = \prod_i \frac{1}{2^n - i}$$

Therefore:

$$\begin{aligned}
&\Pr_{\Omega_X}\left[\omega \in \mathsf{comp}_X(\tau) \mid \omega = (K_H, ., .)\right] \\
=&\Pr_{\pi \leftarrow\$ \text{ Perm}(\{0,1\}^n), F \leftarrow\$ (\mathcal{N} \to \{0,1\}^{l_S})}\left[\forall_i : \eta_{K_H,\pi,F,T^i}(P^i) = C^i\right] \\
=&\prod_i \frac{1}{2^n - i} 2^{-(|P^i|-n)} \\
\geq&\prod_i 2^{-|P^i|} = \Pr_{\Omega_Y}\left[\omega \in \mathsf{comp}_Y(\tau)\right]
\end{aligned}$$

□

Lemma 4.

$$|\rho_X - \rho_Y| \leq (\epsilon + 2^{-n})\binom{q}{2}$$

Proof. Using the H-coefficient technique:

$$\begin{aligned}
&|\rho_X - \rho_Y| \\
\leq&1 - \mathbb{E}_{\tau \sim Y}\left[\min\left(1, \frac{\Pr_{\Omega_X}\left[\omega \in \mathsf{comp}_X(\tau)\right]}{\Pr_{\Omega_Y}\left[\omega \in \mathsf{comp}_Y(\tau)\right]}\right)\right] \\
=&1 - \mathbb{E}_{\tau \sim Y}\left[\min\left(1, \sum_{K_H \in \mathcal{K}_H} \frac{\Pr_{\Omega_X}\left[\omega \in \mathsf{comp}_X(\tau) \wedge \omega = (K_H, ., .)\right]}{\Pr_{\Omega_Y}\left[\omega \in \mathsf{comp}_Y(\tau)\right]}\right)\right]
\end{aligned}$$

by Lemma 3

$$\begin{aligned}
\leq&1 - \mathbb{E}_{\tau \sim Y}\left[\Pr_{K_H \in \mathcal{K}_H}\left[(K_H, \tau) \notin \mathbf{bad}\right]\right] \\
=&\Pr_{\tau \sim Y, K_H \in \mathcal{K}_H}\left[(K_H, \tau) \in \mathbf{bad}\right] \\
\leq&\Pr_{\tau \sim Y, K_H \in \mathcal{K}_H}\left[(K_H, \tau) \in \mathbf{badQ}\right] + \Pr_{\tau \sim Y, K_H \in \mathcal{K}_H}\left[(K_H, \tau) \in \mathbf{badR}\right]
\end{aligned}$$

by Lemma 1 and Lemma 2

$$\leq(\epsilon + 2^{-n})\binom{q}{2}$$

□

5.6 Proof of primary claim

Proof of Theorem 1. To prove this theorem we need a bound on $|\rho_V - \rho_Z|$ where

$$\rho_V \stackrel{\text{def}}{=} \Pr_{K_S \leftarrow\$ \mathcal{K}_S}\left[A^{\mathrm{HBSH}_{K_S}, \mathrm{HBSH}^{-1}_{K_S}} \Rightarrow 1\right]$$
$$\rho_Z \stackrel{\text{def}}{=} \Pr_{\boldsymbol{\pi} \leftarrow\$ \mathrm{Perm}^{\mathcal{T}}(\mathcal{M})}\left[A^{\boldsymbol{\pi}, \boldsymbol{\pi}^{-1}} \Rightarrow 1\right]$$

$|\rho_X - \rho_Y| \leq (\epsilon + 2^{-n})\binom{q}{2}$ by Lemma 4. Since we forbid pointless queries, $|\rho_Y - \rho_Z| \leq 2^{-n}\binom{q}{2}$ by Halevi and Rogaway's PRP-RND lemma ([HR03], appendix C, lemma 6).

To bound $|\rho_V - \rho_X|$ we introduce a stepping stone. Let $\bar{\eta}_F \stackrel{\text{def}}{=} \eta_{K_H, E_{K_E}, F}$ where E is a block cipher and $K_E||K_H|| \ldots = F(\lambda)$. Define

$$\rho_W \stackrel{\text{def}}{=} \Pr_{F \leftarrow\$ (\mathcal{N} \to \{0,1\}^{l_S})}\left[A^{\bar{\eta}_F, \bar{\eta}_F^{-1}} \Rightarrow 1\right]$$

Note that $\mathrm{HBSH}_{K_S} = \bar{\eta}_{S_{K_S}}$, so distinguishing ρ_V and ρ_W is just distinguishing the substitution of a PRF for a random function. Including the key schedule, the attacker distinguishing ρ_V and ρ_W makes at most $q + 1$ queries on the stream cipher or random function respectively, and uses at most $|K_E| + |K_H| + q(l_M - n)$ bits of the output; by a standard substitution argument $|\rho_V - \rho_W| \leq \mathsf{Adv}^{\mathrm{prf}}_S(q + 1, |K_E| + |K_H| + q(l_M - n), t')$ where $t' = t + \mathcal{O}(q(l_T + l_M))$.

The differences between ρ_W and ρ_X are the use of a block cipher in place of a random permutation, and the use of $F(\lambda)$ to determine K_E and K_H. Since F is a random function and $F(\lambda)$ is used only here, this is equivalent to choosing them at random; again by a substitution argument we have that $|\rho_W - \rho_X| \leq \mathsf{Adv}^{\pm\mathrm{prp}}_E(q, t')$.

Theorem 1 follows by summing these bounds:
$|\rho_V - \rho_Z| \leq |\rho_V - \rho_W| + |\rho_W - \rho_X| + |\rho_X - \rho_Y| + |\rho_Y - \rho_Z|$. □

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.

[Arc18] Scott Arciszewski. *XChaCha: eXtended-nonce ChaCha and AEAD-XChaCha20-Poly1305*. Internet-Draft draft-arciszewski-xchacha-02. IETF Secretariat, Oct. 2018. URL: http://www.ietf.org/internet-drafts/draft-arciszewski-xchacha-02.txt.

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

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

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

[Ber05a] Daniel J. Bernstein. *Cache-timing attacks on AES*. 2005. URL: https://cr.yp.to/antiforgery/cachetiming-20050414.pdf (visited on 10/17/2018).

[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.

[Bla+99] J. Black et al. "UMAC: Fast and Secure Message Authentication". In: *Advances in Cryptology — CRYPTO' 99*. Ed. by Michael Wiener. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 216–233. ISBN: 978-3-540-48405-9. DOI: 10.1007/3-540-48405-1_14. URL: https://fastcrypto.org/umac/umac_proc.pdf.

[BR99] Mihir Bellare and Phillip Rogaway. "On the Construction of Variable-Input-Length Ciphers". In: *Fast Software Encryption: 6th International Workshop, 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://ia.cr/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, 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, 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://ia.cr/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://ia.cr/2007/028.

[CS14] Shan Chen and John Steinberger. "Tight Security Bounds for Key-Alternating Ciphers". In: *Advances in Cryptology – EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 327–350. ISBN: 978-3-642-55220-5. DOI: 10.1007/978-3-642-55220-5_19. URL: https://ia.cr/2013/222.

[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://ia.cr/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://ia.cr/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://ia.cr/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://ia.cr/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.

[Kro00] Theodore Dennis Krovetz. “Software-optimized Universal Hashing and Message Authentication”. PhD thesis. 2000. ISBN: 0-599-94329-7. URL: https://fastcrypto.org/umac/.

[Kro06] T. Krovetz. *UMAC: Message Authentication Code using Universal Hashing*. RFC 4418. RFC Editor, Mar. 2006. URL: https://www.rfc-editor.org/rfc/rfc4418.txt.

[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://ia.cr/2007/298.

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

[NIS01] National Institute of Standards and Technology. *Advanced Encryption Standard (AES)*. 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.

[Pat09] Jacques Patarin. "The "Coefficients H" Technique". In: *Selected Areas in Cryptography*. Ed. by Roberto Maria Avanzi, Liam Keliher, and Francesco Sica. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 328–345. ISBN: 978-3-642-04159-4. DOI: 10.1007/978-3-642-04159-4_21. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.702.3488.

[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://ia.cr/2007/317.

[Sar09] Palash Sarkar. *Tweakable Enciphering Schemes From Stream Ciphers With IV*. Tech. rep. 2009. URL: https://ia.cr/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: 10.1016/j.ipl.2011.06.014. URL: https://ia.cr/2009/216.

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

[Tou19] Charlotte de la Tour. *Le langage des fleurs*. Garnier Frères, 1819.

[Vai17] Loup Vaillant. *monocypher.c*. 2017. URL: https://github.com/LoupVaillant/Monocypher/blob/

c8e4340a579fcf3a785539bec36399ec04319126/src/monocypher.c (visited on 12/25/2017).

[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.

A ϵAΔU functions for HBSH

Adiantum and HPolyC are identical except for the choice of ϵAΔU hash function $H_{K_H}(T, L)$. In each case the value of ϵ depends on bounds on $|T|$ and $|L|$. If queries to HBSH are bounded to a maximum tweak and plaintext/ciphertext length of $|T| \leq l_T$, $|P|, |C| \leq l_M$ then the bounds on queries to H will be $|T| \leq l_T$, $|L| \leq l_L = l_M - n$.

A.1 Notation

$\text{int} : \{0,1\}^* \to \mathbb{Z}$ is the standard little-endian map (ie $\text{int}(\lambda) = 0$, $\text{int}(0||X) = 2\,\text{int}(X)$, $\text{int}(1||X) = 1 + 2\,\text{int}(X)$), and $X = \text{fromint}_l(y)$ is the unique l-bit sequence such that $\text{int}(X) \equiv y \pmod{2^l}$.

For both Adiantum and HPolyC, the output group for which the ϵAΔU property applies is $\mathbb{Z}/2^{128}\mathbb{Z}$, so we define

$$x \boxplus y = \text{fromint}_{128}(\text{int}(x) + \text{int}(y))$$
$$x \boxminus y = \text{fromint}_{128}(\text{int}(x) - \text{int}(y))$$

A.2 Poly1305

[Ber05b] uses polynomials over the finite field $\mathbb{Z}/(2^{130}-5)\mathbb{Z}$ to define a function we call $\text{Poly1305} : \{0,1\}^{128} \times \{0,1\}^* \to \{0,1\}^{128}$, and proves in Theorem 3.3 that it is ϵAΔU: for any $g \in \{0,1\}^{128}$ and any distinct messages M, M' where $|M|, |M'| \leq l$, $\Pr_{K_H \xleftarrow{\$} \{0,1\}^{128}}[H_{K_H}(M') \boxminus H_{K_H}(M) = g] \leq 2^{-103} \lceil l/128 \rceil$. In that paper this function is used to build a MAC based on AES, while in RFC 7539 [NL15] it's used to build an AEAD mode based on ChaCha20. Note that 22 bits of the 128-bit key are zeroed before use, so every key is equivalent to 2^{22} keys and the effective keyspace is 2^{106}.

Many Poly1305 libraries take parameters $K_H||g, M$ and return $g \boxplus \mathrm{Poly1305}_{K_H}(M)$; where subtraction is needed we suggest using bitwise inversion and the identity $g \boxminus g' = \neg((\neg g) \boxplus g')$.

A.3 HPolyC hashing

HPolyC is the HBSH construction that the first revision of this paper presented, which used Poly1305 together with an injective encoding function. It is simple, fast, and key agile. We require that $|T| < 2^{32}$ and define

$$H_{K_H}(T, L) = \mathrm{Poly1305}_{K_H}(\mathrm{pad}_{128}(\mathrm{int}_{32}(|T|)||T)||L)$$

Thus if for all queries $|T| \leq l_T$ and $|L| \leq l_L$ then:

$$\epsilon = 2^{-103}(\lceil(32 + l_T)/128\rceil + \lceil l_L/128\rceil)$$

A.4 NH

We define a word size $w = 32$, a stride $s = 2$, a number of rounds $r = 4$ and an input size $u = 8192$ such that $2sw$ divides u.

NH [Bla+99; Kro00; Kro06] is then defined over message lengths divisible by $2sw = 128$ and takes a $u + 2sw(r-1) = 8576$-bit key, processing the message in u-bit chunks to produce an output of size $2rw\lceil|M|/u\rceil$; we call this ratio $u/2rw = 32$ the "compression ratio".

procedure NH(K, M)
  $h \leftarrow \lambda$
  while $M \neq \lambda$ **do**
    $l \leftarrow \min(|M|, u)$
    for $i \leftarrow 0, 2sw, \ldots, 2sw(r-1)$ **do**
      $p \leftarrow 0$
      for $j \leftarrow 0, 2sw, \ldots, l - 2sw$ **do**
        for $k \leftarrow 0, w, \ldots, w(s-1)$ **do**
          $a_0 \leftarrow \mathrm{int}(K[i + j + k; w])$
          $a_1 \leftarrow \mathrm{int}(K[i + j + k + sw; w])$
          $b_0 \leftarrow \mathrm{int}(M[j + k; w])$
          $b_1 \leftarrow \mathrm{int}(M[j + k + sw; w])$
          $p \leftarrow p + ((a_0 + b_0) \bmod 2^w)((a_1 + b_1) \bmod 2^w)$
        end for
      end for
      $h \leftarrow h|| \mathrm{fromint}_{2w}(p)$
    end for
    $M \leftarrow M[l; |M| - l]$
  end while

return h
end procedure

This is the largest w where common vector instruction sets (NEON on ARM; SSE2 and AVX2 on x86) natively support the needed $\{0,1\}^w \times \{0,1\}^w \to \{0,1\}^{2w}$ multiply operation. The stride $s = 2$ improves vectorization on ARM32 NEON; larger strides were slower or no faster on every platform we tested on. We choose $r = 4$ since we want $\epsilon = 2^{-rw} \leq 2^{-103}$ to match HPolyC, and a large u for a high compression ratio which reduces the work for the next hashing stage.

NH's speed comes with several inconvenient properties:

- [Kro00] shows that this function is ϵ-almost-Δ-universal, but this holds only over equal-length inputs
- $\epsilon = 2^{-rw}$, but the smallest nonempty output is $2rw$ bits, twice as large as necessary for this ϵ value
- The output size varies with the input size.

A second hashing stage is used to handle these issues.

A.5 Adiantum hashing

For Adiantum we use NH followed by Poly1305 to hash the message. To avoid encoding and padding issues, we hash the message length and tweak with a separate Poly1305 key. In all this takes a $128 + 128 + 8576 = 8832$-bit key.

procedure $\mathrm{H}(K_H, T, L)$
$K_T \leftarrow K_H[0; 128]$
$K_L \leftarrow K_H[128; 128]$
$K_N \leftarrow K_H[256; 8576]$
$H_T \leftarrow \mathrm{Poly1305}_{K_T}(\mathrm{fromint}_{128}(|L|)||T)$
$H_L \leftarrow \mathrm{Poly1305}_{K_L}(\mathrm{NH}_{K_N}(\mathrm{pad}_{128}(L)))$
return $H_T \boxplus H_L$
end procedure

For distinct pairs $(T, L) \neq (T', L')$, we have that if $|L| \neq |L'|$ or $T \neq T'$, then the $128 + |T|$-bit input to Poly1305 with key K_T will differ. Otherwise $|L| = |L'|$ but $L \neq L'$; per [Kro00] the probability NH will compress these to the same value is at most 2^{-128}. If they do not collide, the $256\lceil |L|/8192 \rceil$-bit input to Poly1305 with key K_L will differ. Since the sum of two ϵAΔU functions with independent keys is also ϵAΔU, if for all queries $|T| \leq l_T$ and $|L| \leq l_L$ then this composition is ϵAΔU, with:

$$\begin{aligned}\epsilon &= 2^{-128} + 2^{-103} \lceil \max(128 + l_T, 256 \lceil l_L/8192 \rceil)/128 \rceil \\ &= 2^{-128} + 2^{-103} \max(1 + \lceil l_T/128 \rceil, 2 \lceil l_L/8192 \rceil)\end{aligned}$$

If we limit our Adiantum attacker to at most q queries each of which uses a tweak of length at at most l_T and a plaintext/ciphertext of length at most l_M, then by Theorem 1 their distinguishing advantage is therefore at most:

$$\begin{aligned}&(3(2^{-128}) + 2^{-103} \max(1 + \lceil l_T/128 \rceil, 2 \lceil (l_M - 128)/8192 \rceil)) \binom{q}{2} \\ &+ \mathsf{Adv}^{\pm\mathrm{prp}}_{E_{K_E}}(q, t') \\ &+ \mathsf{Adv}^{\mathrm{prf}}_{S_{K_S}}(8832 + q(l_M - 128), t')\end{aligned}$$

Assuming that the block and stream ciphers are strong, this is dominated by the term for internal collisions: $2^{-103} \max(1 + \lceil l_T/128 \rceil, 2 \lceil (l_M - 128)/8192 \rceil) \binom{q}{2}$. How many messages can be safely encrypted with the mode will therefore vary with message and tweak length. For example, if Adiantum is used to encrypt 4KiB sectors with 32 byte tweaks, then $\mathrm{Poly1305}_{K_L}$ processes 8 blocks, and the above is approximately $2^{-101} q^2$. With these message and tweak lengths we would recommend encrypting no more than 2^{55} bytes with a single key. Generating the ciphertext to mount such an attack could be very time-consuming, and this is work that can only be done on the device that has the key; extrapolating from performance figures in section 4:

Bytes of ciphertext	Advantage	Time on device (single-threaded)
512GiB	2^{-47}	80 minutes
2^{55}	2^{-15}	11 years
2^{59}	0.8%	175 years

Version: `34654a3ff8c62a8c172c21bb9299de843fef809b`