Adiantum: length-preserving encryption for entry-level processors

Paul Crowley and Eric Biggers

Google LLC

{paulcrowley,ebiggers}@google.com

Abstract. We present HBSH, a simple construction for tweakable length-preserving encryption which 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.

Keywords: super-pseudorandom permutation · variable input length · tweakable encryption · disk encryption

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 adversary 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 (VIL) primitive presented to the AES contest. A key innovation was the idea of a “spice”, which was later formalized as a “tweak” in [LRW02]. Mercy [Cro01] was a tweakable length-preserving primitive designed for sector encryption and 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 VIL cipher from a hash function and a stream cipher (see also BEAST [Luc96a]).
[Luc96b] shows that a universal function suffices for the first round, which [NR99] extends to a 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. SPC mode [BD99] extends this to an SPRP, [HR03] gives the concrete security definition for a tweakable, variable-length, length-preserving SPRP, and defines CMC mode, a two-pass mode with a quadratic security bound. EME mode [HR04] is similar but parallelizable, while EME* mode [Hal05] extends EME mode to handle messages that are not a multiple of the block cipher size. PEP [CS06], TET [Hal07], and HEH [Sar07] have a mixing layer on either side of an ECB layer.

XCB [MF07] is a block-cipher based unbalanced three-round Feistel network with an $\epsilon$-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. HCTR [WFW05; CN08; Kum18], HCH [CS08], HSE [MM07], 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 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 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 GF(2^{128}) polynomial hash. Since these are the operations that act on the bulk of the data in a disk-sector-sized message, 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 $\epsilon$-almost-$\Delta$-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 HPolyC, which does not use NH. HPolyC is 30% slower on large messages, but simpler and much more key agile.
procedure HBSHEncrypt(T, P)
\[ P_L \parallel P_R \leftarrow P \]
\[ P_M \leftarrow P_R \oplus H_{KH}(T, P_L) \]
\[ C_M \leftarrow E_{KE}(P_M) \]
\[ C_L \leftarrow P_L \oplus S_{KS}(C_M)[0;|P_L|] \]
\[ C_R \leftarrow C_M \oplus H_{KH}(T, C_L) \]
\[ C \leftarrow C_L \parallel C_R \]
return C
end procedure

procedure HBSHDecrypt(T, C)
\[ C_L \parallel C_R \leftarrow C \]
\[ C_M \leftarrow C_R \oplus H_{KH}(T, C_L) \]
\[ P_L \leftarrow C_L \oplus S_{KS}(C_M)[0;|C_L|] \]
\[ P_M \leftarrow E_{KE}^{-1}(C_M) \]
\[ P_R \leftarrow P_M \oplus H_{KH}(T, P_L) \]
\[ P \leftarrow P_L \parallel P_R \]
return P
end procedure

Figure 1: HBSH

Figure 2: Pseudocode for HBSH; \( P_R, P_M, C_M, C_R \) are \( n \) bits long

2 Specification

Notation: Partial application is implicit; if we define \( f : A \times B \rightarrow C \) and \( a \in A \) then \( f_a : B \rightarrow C \) and if \( f_a^{-1} \) exists then \( f_a^{-1}(f_a(b)) = b \).

- \(|X|\): length of \( X \in \{0,1\}^* \) in bits
- \( \lambda \): the empty string \(|\lambda| = 0\)
- \( || \): bitstring concatenation
- \( Y[a; l] \): the subsequence of \( Y \) of length \( l \) starting at the 0-based index \( a \).
- \( \text{pad}_l(X) = X \parallel 0^v \) where \( v \) is the least integer \( \geq 0 \) such that \( l \) divides \(|X| + v\)
- \( \text{int} : \{0,1\}^* \rightarrow \mathbb{Z} \): the standard little-endian map such that \( \text{int}(\lambda) = 0 \), \( \text{int}(0 \parallel X) = 2 \text{int}(X) \), \( \text{int}(1 \parallel X) = 1 + 2 \text{int}(X) \)
- \( \text{bin}_l(y) = X \): the unique \( l \)-bit sequence such that \( \text{int}(X) \equiv y \pmod{2^l} \)
- \( n, l_S \): parameters which depend on the primitives from which HBSH is built; for Adiantum and HPolyC, \( n = 128 \) and \( l_S = 2^{73} \)
- \( \mathcal{T} \): the space of tweaks, which depends on the hash function used
- \( \mathcal{M} = \bigcup_{i=0}^{l_S + n} \{0,1\}^i \): the space of plaintexts and ciphertexts
- \( \mathcal{L} = \bigcup_{i=0}^{l_S} \{0,1\}^i \), \( \mathcal{R} = \{0,1\}^n \): messages are processed in two parts, \( \mathcal{L} \times \mathcal{R} \)
- \( H : \mathcal{KH} \times \mathcal{T} \times \mathcal{L} \rightarrow \mathcal{R} \): two-argument hash function with keyspace \( \mathcal{KH} \)
- \( \oplus, \sqcup : \mathcal{R} \times \mathcal{R} \rightarrow \mathcal{R} \): group operations
- \( E : \mathcal{KE} \times \mathcal{R} \rightarrow \mathcal{R} \): \( n \)-bit block cipher with key space \( \mathcal{KE} \)
• $S : \mathcal{K}_S \times \mathcal{N} \to \{0, 1\}^{l_S}$: stream cipher with key space $\mathcal{K}_S$ and nonce space $\mathcal{N}$

• HBSH: 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 part of $n$ bits and a left-hand part with the remainder of the input, and applies an unbalanced Feistel network.

Hash: $H$ is an $\epsilon$-almost-$\Delta$-universal ($\epsilon$-$\Delta U$) function (as defined in Subsection 5.2) yielding a group element represented as an $n$-bit string. $\oplus$ represents addition in a group which depends on the hash function, and $\ominus$ subtraction.

Adiantum and HPolyC differ only in their choice of hash function. HPolyC is based on Poly1305, while Adiantum uses both Poly1305 and NH; specifically little-endian NH$^T$[256; 32; 4] with a stride of 2 for fast vectorization. In both cases, the group used for $\oplus$ and $\ominus$ is $\mathbb{Z}/2^{128}\mathbb{Z}$. The value of $\epsilon$ depends on bounds on the input lengths. We defer full details to Section 6.

Block cipher: The block cipher $E$ 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 $n = 128$ and $\mathcal{K}_E = \{0, 1\}^{256}$.

Stream cipher: $S$ 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 $\lambda$, which is distinct from all $n$-bit strings; thus $\{\lambda\} \cup \mathcal{R} \subseteq \mathcal{N}$.

Adiantum and HPolyC use the XChaCha12 stream cipher. The ChaCha [Ber08a] stream ciphers take a 64-bit nonce, and RFC7530 [NL15] proposes a ChaCha20 variant with a 96-bit nonce, but we need a 128-bit nonce. The XSal20 construction [Ber11] proposed for Salsa20 [Ber08b; Ber06] extends the nonce to 192 bits, and applies straightforwardly to ChaCha [Arc18; Vai18; Den18]. We then construct a function that takes a variable-length nonce of up to 191 bits by padding with a 1 followed by zeroes: $S_{K_S}(C_M) = XChaCha12_{K_S}(pad_{192}(C_M||1))$ and $\mathcal{N} = \bigcup_{i=0}^{194} \{0, 1\}^i$. The maximum output length $l_S = 2^{73}$ and keyspace $\mathcal{K}_S = \{0, 1\}^{256}$.

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.

A second, functional definition of HBSH is given in Subsection 5.1.

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. $\epsilon$-$\Delta U$ functions are the fastest options for reading the plaintext in a cryptographically useful way, and stream ciphers are the fastest options for modifying it. $\epsilon$-$\Delta U$s 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 messages in non-round sizes; many VIL modes need extra wrinkles akin to ciphertext stealing to handle the case where the message is not a multiple of the block size of the underlying block cipher.

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 message, 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 

$$C = E_{K_E}(P \oplus H_{K_H}(T, \lambda)) \oplus H_{K_H}(T, \lambda)$$

as per Subsection 3.1 of [LRW02]. 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.

Components: It’s unusual for a construction to require more than two distinct primitive components. More commonly, a hash-XOR-hash mode uses the block cipher to build a stream cipher (eg using CTR mode [LWR00]) and also uses it directly on the narrow side of the message. 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 $\epsilon$-$\Delta U$ 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 message, 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 adversary 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.

Stream cipher: Users are highly sensitive to the performance of disk encryption; an extra microsecond decrypting the contents of a sector can mean many users forgoing encryption altogether. eBACS [BL18] tests a wide variety of stream ciphers on a wide variety of architectures; ChaCha12 is consistently one of the fastest options for the “armeabi” (32-bit ARM) architecture. ChaCha and its predecessor Salsa20 have seen intense cryptanalysis in the decade or so since publication [Cro06; Fis+06; Tsu+07; Aum+08; IKM11; Ish12; Shi+13; MPM15; Mai16; CM16; DS17; CM17; DS18]; the best attack breaks 7 rounds, a landmark reached with [Aum+08] in 2008. Each round greatly increases the difficulty of attack. We therefore feel confident selecting the 12-round variant as giving good confidence in security while minimizing the cost to users.

Hash function: Since the $\epsilon$-$\Delta U$ is run twice over the bulk of the message, its speed is especially crucial for large messages. One of the fastest such functions in software is NH, and it’s also appealingly simple; however as discussed in Subsection 6.3 it generally has to be combined with a second hashing stage, and for this purpose we use Poly1305. The 1KiB input size we specify for NH means that a simple, portable implementation of Poly1305 can be used without a great cost in speed; in contrast, for HPolyC a vectorized Poly1305 implementation is important. 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 messages 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 messages may instead use HPolyC, which uses Poly1305 directly. 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 requires one extra call to the ChaCha permutation for each new nonce. No extra work is required for differing message or tweak lengths for either Adiantum or HPolyC.
**Table 1:** Performance on ARM Cortex-A7

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Cycles per byte (4096-byte sectors)</th>
<th>Cycles per byte (512-byte sectors)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NH</td>
<td>1.3</td>
<td>1.4</td>
</tr>
<tr>
<td>Poly1305</td>
<td>2.9</td>
<td>3.3</td>
</tr>
<tr>
<td>ChaCha8</td>
<td>5.1</td>
<td>5.2</td>
</tr>
<tr>
<td>ChaCha12</td>
<td>7.1</td>
<td>7.2</td>
</tr>
<tr>
<td>Adiantum-XChaCha8-AES</td>
<td>8.5</td>
<td>13.2</td>
</tr>
<tr>
<td><strong>Adiantum-XChaCha12-AES</strong></td>
<td><strong>10.6</strong></td>
<td><strong>15.8</strong></td>
</tr>
<tr>
<td>ChaCha20</td>
<td>11.2</td>
<td>11.3</td>
</tr>
<tr>
<td>HPolyC-XChaCha8-AES</td>
<td>11.5</td>
<td>16.5</td>
</tr>
<tr>
<td><strong>HPolyC-XChaCha12-AES</strong></td>
<td><strong>13.6</strong></td>
<td><strong>18.7</strong></td>
</tr>
<tr>
<td>Adiantum-XChaCha20-AES</td>
<td>14.7</td>
<td>20.2</td>
</tr>
<tr>
<td>Speck128/128-XTS</td>
<td>15.0</td>
<td>16.1</td>
</tr>
<tr>
<td>Speck128/256-XTS</td>
<td>15.8</td>
<td>16.9</td>
</tr>
<tr>
<td>HPolyC-XChaCha20-AES</td>
<td>17.8</td>
<td>23.4</td>
</tr>
<tr>
<td>NOEKEON-XTS</td>
<td>26.9</td>
<td>27.9</td>
</tr>
<tr>
<td>XTEA-XTS</td>
<td>28.7</td>
<td>29.7</td>
</tr>
<tr>
<td>AES-128-XTS (encryption)</td>
<td>36.1</td>
<td>37.2</td>
</tr>
<tr>
<td>AES-128-XTS (decryption)</td>
<td>42.7</td>
<td>43.9</td>
</tr>
<tr>
<td>AES-256-XTS (encryption)</td>
<td>48.9</td>
<td>50.5</td>
</tr>
<tr>
<td>AES-256-XTS (decryption)</td>
<td>58.6</td>
<td>60.1</td>
</tr>
</tbody>
</table>

**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” part 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

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. 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 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
Table 2: Implementations

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Source</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChaCha</td>
<td>Linux v4.20-rc1</td>
<td>chaacha20-neon-core.S, modified to support ChaCha8 and ChaCha12</td>
</tr>
<tr>
<td>Poly1305</td>
<td>OpenSSL 1.1.0h</td>
<td>poly1305-armv4.S, modified to precompute key powers just once per key</td>
</tr>
<tr>
<td>AES</td>
<td>Linux v4.17</td>
<td>aes-cipher-core.S, modified to prefetch lookup tables</td>
</tr>
<tr>
<td>AES-XTS</td>
<td>Linux v4.17</td>
<td>aes-neonzs-core.S (bit-sliced)</td>
</tr>
<tr>
<td>Speck128/256-XTS</td>
<td>Linux v4.17</td>
<td>speck-neonzs-core.S</td>
</tr>
<tr>
<td>NOEKEON-XTS</td>
<td>ours</td>
<td></td>
</tr>
<tr>
<td>XTEA-XTS</td>
<td>ours</td>
<td></td>
</tr>
</tbody>
</table>

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 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.

5 Security of HBSH

Assuming the security of the underlying block and stream ciphers, we show here that HBSH has an advantage bound that grows quadratically with the number of queries.

5.1 Definition of HBSH

We provide here an equivalent definition of HBSH in functional form. Where a parameter is given as $L||R$ we have that $L \in \mathcal{L}$, $R \in \mathcal{R}$, $L||R \in \mathcal{M}$.

$$\xi : K_H \times T \times \mathcal{M} \to \mathcal{R}$$

$$\xi_{K_H}(T, L||R) \overset{\text{def}}{=} R \oplus H_{K_H}(T, L)$$
Adiantum: length-preserving encryption for entry-level processors

\[ \phi : K_H \times \mathcal{T} \times \mathcal{M} \to \mathcal{M} \]

\[ \phi_{K_H, T}(L||R) \overset{\text{def}}{=} L||H_{K_H}(T, L||R) \]

\[ \phi_{K_H, T}^{-1}(L||R) = L||(R \oplus H_{K_H}(T, L)) \]

\[ \theta : \text{Perm}(\mathcal{R}) \times (\mathcal{N} \to \{0, 1\}^{|s|}) \times \mathcal{T} \times \mathcal{M} \to \mathcal{M} \]

\[ \theta_{\pi, F}(L||R) \overset{\text{def}}{=} (L \oplus F(\pi(R))[0;|L|])\pi(R) \]

\[ \theta_{\pi}^{-1}(L||R) = (L \oplus F(R)[0;|L|])\pi^{-1}(R) \]

\[ \eta : K_H \times \text{Perm}(\mathcal{R}) \times (\mathcal{N} \to \{0, 1\}^{|s|}) \times \mathcal{T} \times \mathcal{M} \to \mathcal{M} \]

\[ \eta_{K_H, \pi, F, T} \overset{\text{def}}{=} \phi_{K_H, T}^{-1} \circ \theta_{\pi, F} \circ \phi_{K_H, T} \]

\[ \tilde{\eta} : (\mathcal{N} \to \{0, 1\}^{|s|}) \times \mathcal{T} \times \mathcal{M} \to \mathcal{M} \]

\[ \tilde{\eta}_E \overset{\text{def}}{=} \eta_{K_H, E_{K_H}, F} \quad \text{where } K_E \| K_H \| \ldots = F(\lambda) \]

\[ \text{HBSH} : K_S \times \mathcal{T} \times \mathcal{M} \to \mathcal{M} \]

\[ \text{HBSH}_{K_S} \overset{\text{def}}{=} \tilde{\eta}_{K_S} \]

### 5.2 Security definitions

**Hash function:** The hash function \( H \) must be \( \epsilon \)-almost-\( \Delta \)-universal (\( \epsilon \)-\( \Delta \)U) for some \( \epsilon \) [Sti95]: for any \( g \in \mathcal{R} \) and any two distinct messages \( (T, L) \neq (T', L') \):

\[
\Pr_{K \leftarrow K_H} [H_K(T, L) \oplus H_K(T', L') = g] \leq \epsilon
\]

Given bounds on the lengths of \( T \) and \( L \), the value of \( \epsilon \) for the hash function used in HPolyC is given in Subsection 6.2, and for Adiantum in Subsection 6.4.

**Block cipher:** The block cipher \( E \) must be a super-pseudorandom permutation [Bel+97]:

\[
\text{Adv}_{E}^{\pm \text{prp}}(A) \overset{\text{def}}{=} \left| \Pr_{K \leftarrow K_E} [A^{E_{K, K^{-1}}_E} \Rightarrow 1] - \Pr_{\pi \leftarrow \text{Perm}(\mathcal{R})} [A^{\pi^{-1}} \Rightarrow 1] \right|
\]

\[
\text{Adv}_{E}^{\pm \text{prp}}(q, t) \overset{\text{def}}{=} \max_{A \in \mathcal{A}(q, t)} \text{Adv}_{E}^{\pm \text{prp}}(A)
\]

where \( A \) is an adversary, \( \text{Perm}(S) \) 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.

**Stream cipher:** Our definition is related to the definition of a PRF in [Bel+97], but because we model the stream cipher \( S \) as a pseudorandom function with a very long output, we bound the adversary not only in how many queries they make, but also in how many bits they read in total. Thus, a query consists of a pair \((N, l_q) \in \mathcal{N} \times \mathbb{N}\) where \( 0 < l_q \leq l_S \), and the response is \( S_K(N)[0; l_q] \) or \( F(N)[0; l_q] \); we cap the sum of \( l_q \) values
across queries.
\[
\text{Adv}^\text{sc}\left( A \right) \overset{\text{def}}{=} \left| \Pr_{K \leftarrow K_S} \left[ A^{S_K(\cdot)}[0;] \Rightarrow 1 \right] - \Pr_{F \leftarrow \mathcal{N}^\rightarrow\{0,1\}^l} \left[ A^{F(\cdot)}[0;] \Rightarrow 1 \right] \right|
\]
\[
\text{Adv}^\text{sc}\left( q, l, t \right) \overset{\text{def}}{=} \max_{A \in \mathcal{A}(q,l,t)} \text{Adv}^\text{sc}\left( A \right)
\]

where \( A \) is an adversary, \( \mathcal{N}^\rightarrow\{0,1\}^l \) denotes the set of all functions from \( \mathcal{N} \) to \( \{0,1\}^l \), and \( \mathcal{A}(q,l,t) \) is the set of all adversaries that make at most \( q \) queries such that \( \sum l_q \leq l \), and which take at most \( t \) time.

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

\[
\text{Perm}^T(\mathcal{M}) \overset{\text{def}}{=} \{ \pi \in \text{LP}^T(\mathcal{M}) : \forall T, \pi_T \text{ is a bijection} \}
\]

Per [HR03], for a tweakable, variable-input-length, super-pseudorandom permutation \( E : K \times T \times \mathcal{M} \rightarrow \mathcal{M} \) the distinguishing advantage of an adversary \( A \) is:
\[
\text{Adv}^\pm\text{prp}_E\left( A \right) \overset{\text{def}}{=} \left| \Pr_{K \leftarrow K} \left[ A^{E_K \cdot E_K^{-1}}[\cdot] \Rightarrow 1 \right] - \Pr_{\pi \leftarrow \text{Perm}^T(\mathcal{M})} \left[ A^{\pi^{-1}}[\cdot] \Rightarrow 1 \right] \right|
\]
\[
\text{Adv}^\pm\text{prp}\left( q, l_T, l_M, t \right) \overset{\text{def}}{=} \max_{A \in \mathcal{A}(q,l_T,l_M,t)} \text{Adv}^\pm\text{prp}\left( A \right)
\]

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.

### 5.3 Primary claim

**Theorem 1.** Where HBSH mode is instantiated with hash function \( H \), block cipher \( E \) and stream cipher \( S \), and where \( H \) is \( \epsilon \)-almost-\( \Delta \)-universal for inputs such that \( |T| \leq l_T, |L| \leq l_M - n \), then:
\[
\text{Adv}^\pm\text{prp}_{\text{HBSH}}(q, l_T, l_M, t) \leq (\epsilon + 2(2^{-n})) \left( \frac{q}{2} \right) + \text{Adv}^\text{sc}_S(q+1, |K_E| + |K_H| + q(l_M - n), t') + \text{Adv}^\pm\text{prp}_E(q, t')
\]

where \( t' = t + O(q(l_T + l_M)) \).

This is proven in what follows. First we use the H-coefficient technique to establish Lemma 4, a closely related bound; then in Subsection 5.7 we bridge the gap between this bound and the desired bound.

### 5.4 H-coefficient technique

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

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

- $\Omega_X$: a set of instances we sample fairly from. We write $\Pr_{\omega \leftrightarrow \Omega_X}$ as shorthand for $\Pr_{\omega \leftrightarrow \Omega_X}$.
- $\Upsilon_X$: a map from an instance $\omega \in \Omega_X$ to a tuple of deterministic oracles we can present to the adversary.
- $\rho_X \overset{\text{def}}{=} \Pr_{\Omega_X} \left[ A^{\Upsilon_X(\omega)} \Rightarrow 1 \right]$ where the adversary $A$ is clear from context. As the adversary interacts with the oracles, a transcript $\tau$ of queries and responses is generated.
- $X$: a random variable representing a transcript for $A^{\Upsilon_X(\omega)}$ given $\omega \leftrightarrow \Omega_X$; we write $\tau \leftrightarrow X$ to indicate that $\tau$ is sampled from this distribution.
- $\text{comp}_X$: We write $\omega \in \text{comp}_X(\tau)$ if a transcript $\tau$ is “compatible” with an instance $\omega \in \Omega_X$, ie if given an adversary $A$ that makes those queries, the oracles $\Upsilon_X(\omega)$ make those responses and thus $A^{\Upsilon_X(\omega)}$ produces that transcript.

We have the same for world $Y$ throughout.

We assume a deterministic adversary. This is without loss of generality; if we assume a distribution of adversaries $A \leftrightarrow \mathcal{A}$ then an advantage bound on each of the deterministic adversaries $A$ bounds the advantage of the ensemble $\mathcal{A}$.

Once $\omega$ is sampled, the oracles $\Upsilon_X(\omega)$ are then deterministic; the transcript produced by $A^{\Upsilon_X(\omega)}$ is thus the unique transcript compatible both with adversary $A$ and instance $\omega$. Where a transcript is not compatible with $A$, $\Pr[X = \tau] = \Pr[Y = \tau] = 0$. If either of these is not zero, the transcript is compatible with $A$, and $\Pr[X = \tau] = \Pr_{\Omega_X}[\omega \in \text{comp}_X(\tau)]$ and similarly for $Y$.

The adversary always returns the same result for the same transcript, so its advantage is maximized if it returns 1 exactly when $\Pr[Y = \tau] > \Pr[X = \tau]$. Therefore:

\[
\text{Adv}_X^Y(A) = |\rho_X - \rho_Y| \\
\leq \sum_{\tau \colon \Pr[Y = \tau] > \Pr[X = \tau]} \left( \Pr[Y = \tau] - \Pr[X = \tau] \right) \\
= \sum_{\tau \colon \Pr[Y = \tau] > \Pr[X = \tau]} \Pr[Y = \tau] \left( 1 - \frac{\Pr[X = \tau]}{\Pr[Y = \tau]} \right) \\
= \sum_{\tau \colon \Pr[Y = \tau] > 0} \Pr[Y = \tau] \left( 1 - \min \left( 1, \frac{\Pr[X = \tau]}{\Pr[Y = \tau]} \right) \right) \\
= \mathbb{E}_{\tau \leftrightarrow Y} \left[ 1 - \min \left( 1, \frac{\Pr[X = \tau]}{\Pr[Y = \tau]} \right) \right] \\
= 1 - \mathbb{E}_{\tau \leftrightarrow Y} \left[ \min \left( 1, \frac{\Pr_{\Omega_X}[\omega \in \text{comp}_X(\tau)]}{\Pr_{\Omega_Y}[\omega \in \text{comp}_Y(\tau)]} \right) \right]
\]

where $\mathbb{E}$ is the expected value. With this rearrangement, the only probability distribution we sum over is that of $Y$, which can be more convenient to work with.

### 5.5 Preliminaries

**World $X$**: World $X$ is an idealized form of HBSH which uses a random function and permutation: $\Omega_X \overset{\text{def}}{=} K_H \times \text{Perm}(R) \times (\mathcal{N} \rightarrow \{0,1\}^r)$, and given $(K_H, \pi, F) \in \Omega_X$, $\Upsilon_X(K_H, \pi, F) \overset{\text{def}}{=} (\eta_{K_H, \pi, F}, \eta_{K_H, \pi, F}^{-1})$. 


**World Y**: World Y samples fairly from all possible pairs of tweakable length-preserving functions: \( \Omega_Y \overset{\text{def}}{=} \text{LP}^T(M) \times \text{LP}^T(M) \), so given \((E, D) \in \Omega_Y\), \(TY(E, D) \overset{\text{def}}{=} (E, D)\).

**Transcript**: Our transcript \( \tau \) is a sequence of tuples \((d^i, T^i, P^i, C^i)\) in \([+,-] \times T \times M \times M\) for \(i \in [0, \ldots, q - 1]\). For the \(i\)th 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 \(A(q, l_T, l_M, t)\) 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 \(d^i = +\) then \((T^i, P^i) \neq (T^j, P^j)\), and similarly if \(d^i = -\) then \((T^i, C^i) \neq (T^j, C^j)\).

**Bad events**: We define various classes of “bad event”:

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

Finally we define the disjunction \(\text{bad} \overset{\text{def}}{=} \text{badQ} \cup \text{badR}\).

### 5.6 Lemmas

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

\[
\Pr_{K_H \leftarrow_s K_H} [(K_H, \tau) \in \text{badQ}] \leq \epsilon \left(\frac{q}{2}\right)
\]

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

\[
\xi_{K_H}(T^i, L^i \parallel R^i) = \xi_{K_H}(T^j, L^j \parallel R^j) \\
\Rightarrow R^i \parallel H_{K_H}(T^i, L^i) = R^j \parallel H_{K_H}(T^j, L^j) \\
\Rightarrow H_{K_H}(T^i, L^i) \parallel H_{K_H}(T^j, L^j) = R^i \parallel R^j
\]

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

Where \(d^i = -\), a similar argument applies for \(C^i, C^j\). For an upper bound, we sum across all \(\left(\begin{array}{c} q \\ 2 \end{array}\right)\) pairs \(i, j\).

**Lemma 2.** For any \(K_H \leftarrow_s K_H\),

\[
\Pr_{\tau \leftarrow_s Y} [(K_H, \tau) \in \text{badR}] \leq 2^{-n} \left(\frac{q}{2}\right)
\]
Proof. Assume \(d_i^j = +\) for some pair \(i, j\), and let \(L^i \| R^i = C^i\) and similarly for \(C^j\). Because pointless queries are forbidden, in world \(Y\), conditioning on all prior queries and responses, all possible values of \(C^j\) such that \(|C^j| = |P^i|\) will be equally likely. In particular, even after conditioning on \(L^i\), all values of \(R^i\) are equally likely. Therefore 

\[
\Pr[R^i = R^i \oplus H_{K_H}(T^i, L^i) \oplus H_{K_H}(P^i, L^i)] = 2^{-n}.
\]

Where \(d_i^j = \cdot\), a similar argument applies for \(P^i, P^j\). For an upper bound, we sum across all \(\binom{n}{2}\) pairs \(i, j\). \(\square\)

**Lemma 3.** For any \(K_H \in K_H\) and transcript \(\tau\) such that \(\Pr[Y = \tau] > 0\) and \((K_H, \tau) \not\in \text{bad}\),

\[
\Pr_{\Omega_X} [\omega \in \text{comp}_X(\tau) \mid \omega = (K_H, \ldots)] \geq \Pr_{\Omega_Y} [\omega \in \text{comp}_Y(\tau)\]

*Proof.* In world \(Y\), for any transcript such that \(\Pr[Y = \tau] > 0\), since all queries are distinct, the responses are independent fair random draws of binary strings of the appropriate length, and therefore \(\Pr_{\Omega_Y} [\omega \in \text{comp}_Y(\tau)] = \prod_i 2^{-|P^i|}\).

For world \(X\), let \(P^i_L \| P^i_R = P^i\), \(P^i_M = \xi_{K_H,T^i}(P^i)\) and similarly for \(C^i\). Since \((K_H, \tau) \not\in \text{bad}\) we have that \(P^i_M \neq P^j_M\) and \(C^i_M \neq C^j_M\) for all \(i \neq j\). \((K_H, \pi, F) \in \text{comp}_X(\tau)\) exactly if, for each \(i\):

\[
\begin{align*}
\eta_{K_H, \pi, F, T^i}(P^i) &= C^i \\
&\iff \phi_{K_H,T^i}((\theta_{\pi,F}(\phi_{K_H,T^i}(P^i)))) = C^i \\
&\iff \theta_{\pi,F}(P^i_L \| P^i_M) = C^i_L \parallel C^i_M \\
&\iff P^i_L \oplus F(C^i_M)[0; |P^i|] = C^i_L \parallel \pi(P^i_M) = C^i_M \\
&\iff F(C^i_M)[0; |P^i| - n] = P^i_L \oplus C^i_L \parallel \pi(P^i_M) = C^i_M
\end{align*}
\]

Since \(\pi\) and \(F\) are drawn independently, we can consider these conditions on them separately. For \(F\), since all \(C^i_M\) are distinct, these are once again independent fair random draws of binary strings of the appropriate length:

\[
\Pr_{F \leftrightarrow (N \rightarrow \{0, 1\}^{|\tau|})} [\forall i : F(C^i_M)[0; |P^i| - n] = P^i_L \oplus C^i_L] = \prod_i 2^{-|P^i| - n}
\]

For \(\pi\), again given that all \(P^i_M\) are distinct and all \(C^i_M\) are distinct, we have that

\[
\Pr_{\pi \leftrightarrow \text{Perm}(\mathcal{R})} \left[\pi(P^i_M) = C^i_M \mid \forall 0 \leq i < j : \pi(P^i_M) = C^i_M\right] = \frac{1}{2^n - j}
\]

(we number queries in the range \(i \in [0 \ldots q - 1]\)) and therefore that:

\[
\Pr_{\pi \leftrightarrow \text{Perm}(\mathcal{R})} \left[\forall i : \pi(P^i_M) = C^i_M\right] = \prod_i \frac{1}{2^n - i}
\]

Therefore:

\[
\begin{align*}
\Pr_{\Omega_X} [\omega \in \text{comp}_X(\tau) \mid \omega = (K_H, \ldots)] \\
&= \Pr_{\pi \leftrightarrow \text{Perm}(\mathcal{R}), F \leftrightarrow (N \rightarrow \{0, 1\}^{|\tau|})} [\forall i : \eta_{K_H, \pi, F, T^i}(P^i) = C^i] \\
&= \prod_i \frac{1}{2^n - i} 2^{-|P^i| - n} \\
&\geq \prod_i 2^{-|P^i|} = \Pr_{\Omega_Y} [\omega \in \text{comp}_Y(\tau)] \quad \square
\end{align*}
\]
Lemma 4.
\[ |\rho_X - \rho_Y| \leq (\epsilon + 2^{-n}) \left( \frac{q}{2} \right) \]

Proof. For any transcript \( \tau \) such that \( \Pr[Y = \tau] > 0 \):
\[
\min \left( 1, \frac{\Pr_{\Omega_X}[\omega \in \text{comp}_X(\tau)]}{\Pr_{\Omega_Y}[\omega \in \text{comp}_Y(\tau)]} \right) \\
= \min \left( 1, \frac{\E_{K_H \leftrightarrow K_H} \Pr_{\Omega_X}[\omega \in \text{comp}_X(\tau) | \omega = (K_H, \ldots)]}{\Pr_{\Omega_Y}[\omega \in \text{comp}_Y(\tau)]} \right)
\]
\[
\min (1, E[U]) \geq E[\min (1, U)] \text{ for any real random variable } U, \text{ therefore}
\]
\[
\geq \E_{K_H \leftrightarrow K_H} \left[ \min \left( 1, \frac{\Pr_{\Omega_X}[\omega \in \text{comp}_X(\tau) | \omega = (K_H, \ldots)]}{\Pr_{\Omega_Y}[\omega \in \text{comp}_Y(\tau)]} \right) \right] \\
\geq \Pr_{K_H \leftrightarrow K_H} \left[ \frac{\Pr_{\Omega_X}[\omega \in \text{comp}_X(\tau) | \omega = (K_H, \ldots)]}{\Pr_{\Omega_Y}[\omega \in \text{comp}_Y(\tau)]} \geq 1 \right]
\]
by Lemma 3
\[
\geq \Pr_{K_H \leftrightarrow K_H} [(K_H, \tau) \notin \text{bad}]
\]

Using the H-coefficient technique:
\[
|\rho_X - \rho_Y| \\
\leq 1 - \E_{\tau \leftrightarrow Y} \left[ \min \left( 1, \frac{\Pr_{\Omega_X}[\omega \in \text{comp}_X(\tau)]}{\Pr_{\Omega_Y}[\omega \in \text{comp}_Y(\tau)]} \right) \right] \\
\leq 1 - \E_{\tau \leftrightarrow Y} \left[ \Pr_{K_H \leftrightarrow K_H} [(K_H, \tau) \notin \text{bad}] \right] \\
= \Pr_{\tau \leftrightarrow Y, K_H \leftrightarrow K_H} [(K_H, \tau) \in \text{bad}] \\
\leq \Pr_{\tau \leftrightarrow Y, K_H \leftrightarrow K_H} [(K_H, \tau) \in \text{bad(Q)}] + \Pr_{\tau \leftrightarrow Y, K_H \leftrightarrow K_H} [(K_H, \tau) \in \text{bad(R)}]
\]
by Lemma 1 and Lemma 2
\[
\leq (\epsilon + 2^{-n}) \left( \frac{q}{2} \right)
\]

5.7 Proof of primary claim

Proof of Theorem 1. Copying the definitions of \( \rho_X, \rho_Y \) from Subsection 5.4, we define
\[
\rho_V \overset{\text{def}}{=} \Pr_{K_S \leftrightarrow K_S} \left[ A^{\text{HBSH}_K S, \text{HBSH}_S^{-1}_K} \Rightarrow 1 \right] \\
\rho_W \overset{\text{def}}{=} \Pr_{F \leftrightarrow (N \to \{0,1\}^s)} \left[ A^{\eta_F, \eta_F^{-1}} \Rightarrow 1 \right] \\
\rho_X \overset{\text{def}}{=} \Pr_{\Omega_X} \left[ A^{T_X(\omega)} \Rightarrow 1 \right] \\
= \Pr_{K_H, \pi, F \leftrightarrow \Omega_X} \left[ A^{\delta_{K_H, \pi, F, \eta_{K_H, \pi, F}^{-1}}} \Rightarrow 1 \right] \\
\rho_Y \overset{\text{def}}{=} \Pr_{\Omega_Y} \left[ A^{T_Y(\omega)} \Rightarrow 1 \right] \\
= \Pr_{\pi, D \leftrightarrow \text{LP}^T(M) \times \text{LP}^T(M)} \left[ A^{F, D} \Rightarrow 1 \right] \\
\rho_Z \overset{\text{def}}{=} \Pr_{\pi \leftrightarrow \text{Perm}^T(M)} \left[ A^{\pi, \pi^{-1}} \Rightarrow 1 \right]
\]
Distinguishing $\rho_Y$ and $\rho_W$ is distinguishing the substitution of a stream cipher for a random function. Including the key schedule, the adversary distinguishing $\rho_Y$ and $\rho_W$ makes at most $q + 1$ queries to 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 per [BKR94; Bel97], $|\rho_Y - \rho_W| \leq \text{Adv}^E_\delta(q + 1, |K_E| + |K_H| + q(l_M - n), t')$ where $t' = t + O(q(l_T + l_M))$.

The differences between $\rho_W$ and $\rho_X$ 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 $|\rho_W - \rho_X| \leq \text{Adv}^E_\delta(q, t')$.

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

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

### 6 $\epsilon$-DU functions for HBSH

Adiantum and HPolyC are identical except for the choice of $\epsilon$-DU hash function $H_{KH}(T, L)$. In each case the value of $\epsilon$ 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 $|T| \leq l_T$, $|L| \leq l_L = l_M - n$.

#### 6.1 Poly1305

Both Adiantum and HPolyC make use of the polynomial hash function Poly1305:

$$\text{PolyP} : \mathbb{Z} \times \{0, 1\}^* \rightarrow \mathbb{Z}$$

$$\text{PolyP}(k, \lambda) \overset{\text{def}}{=} 0$$

$$\text{PolyP}(k, M_1 \| M_2) \overset{\text{def}}{=} k(\text{PolyP}(k, M_1) + \text{int}(M_2 \| 1)) \mod 2^{130} - 5$$

where 128 divides $|M_1|$, $0 < |M_2| \leq 128$

$$\text{PolyMask} \overset{\text{def}}{=} 128 \| 0^6 \| 1^{26} \| 0^6 \| 1^{26} \| 0^6 \| 1^{26} \| 0^4$$

$$\text{Poly1305} : \{0, 1\}^{128} \times \{0, 1\}^* \rightarrow \{0, 1\}^{128}$$

$$\text{Poly1305}_K(M) \overset{\text{def}}{=} \text{bin}_{128}(\text{PolyP}(\text{int}(K \land \text{PolyMask}), M))$$

where $\land$ denotes bitwise AND. The output group for which the $\epsilon$-DU property applies is $\mathbb{Z}/2^{128}\mathbb{Z}$, so we define

$$x \oplus y \overset{\text{def}}{=} \text{bin}_{128}(\text{int}(x) + \text{int}(y))$$

$$x \otimes y \overset{\text{def}}{=} \text{bin}_{128}(\text{int}(x) - \text{int}(y))$$

[Ber05b] proves in Theorem 3.3 that this function is $\epsilon$-DU; for any $g \in \{0, 1\}^{128}$ and any distinct messages $M, M'$ where $|M|, |M'| \leq l$:

$$\text{Pr}_{K \leftarrow \{0, 1\}^{128}}[\text{Poly1305}_K(M) \oplus \text{Poly1305}_K(M') = g] \leq 2^{-103}[l/128]$$

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. Since 22 bits of the 128-bit key are zeroed before use, every key is equivalent to $2^{22} - 1$ other keys and the effective keyspace is $2^{106}$. 
6.2 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.

\[
\mathcal{T} = \bigcup_{i=0}^{2^{32} - 1} \{0, 1\}^i
\]

\[
H_{K_H}(T, L) = \text{Poly1305}_{K_H}(\text{pad}_{128}(\text{int}_{32}(|T|)||T)||L)
\]

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

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

6.3 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 \(2rw \lfloor |M|/u \rfloor\); we call this ratio \(u/2rw = 32\) the “compression ratio”.

\[
\text{procedure NH}(K, M) \\
\begin{align*}
  h &\leftarrow \lambda \\
  \text{while } M \neq \lambda \text{ do} \\
  &l \leftarrow \min (|M|, u) \\
  &\text{for } i \leftarrow 0, 2sw, \ldots, 2sw(r - 1) \text{ do} \\
  &\quad p \leftarrow 0 \\
  &\quad \text{for } j \leftarrow 0, 2sw, \ldots, l - 2sw \text{ do} \\
  &\quad \quad \text{for } k \leftarrow 0, w, \ldots, w(s - 1) \text{ do} \\
  &\quad \quad \quad a_0 \leftarrow \text{int}(K[i + j + k; w]) \\
  &\quad \quad \quad a_1 \leftarrow \text{int}(K[i + j + k + sw; w]) \\
  &\quad \quad \quad b_0 \leftarrow \text{int}(M[j + k; w]) \\
  &\quad \quad \quad b_1 \leftarrow \text{int}(M[j + k + sw; w]) \\
  &\quad \quad \quad p \leftarrow p + ((a_0 + b_0) \mod 2^w)((a_1 + b_1) \mod 2^w) \\
  &\quad \text{end for} \\
  &\quad \text{end for} \\
  &h \leftarrow h||\text{bin}_{2w}(p) \\
  &\text{end for} \\
  &M \leftarrow M[l; |M| - l] \\
  &\text{end while} \\
  &\text{return } h \\
\end{align*}
\]

This is the largest \(w\) where common vector instruction sets (NEON on ARM; SSE2 and AVX2 on x86) natively support the needed \(\{0, 1\}^w \times \{0, 1\}^w \rightarrow \{0, 1\}^{2w}\) 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 \(\epsilon = 2^{-103} \leq 2^{-103}\) 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 \( \epsilon \)-almost-\( \Delta \)-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 \( \epsilon \) value

• The output size varies with the input size.

A second hashing stage is used to handle these issues.

6.4 Adiantum hashing

For Adiantum we use NH followed by Poly1305 to hash the message. Our theorems assume \( \mathcal{T} \) is finite, so we somewhat arbitrarily set \( \mathcal{T} = \mathcal{L} = \bigcup_{i=0}^{l_T} \{0,1\}^i \). 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.

\[
\text{procedure } H(K_H, T, L) \\
\begin{align*}
K_T &\leftarrow K_H[0; 128] \\
K_L &\leftarrow K_H[128; 128] \\
K_N &\leftarrow K_H[256; 8576] \\
H_T &\leftarrow \text{Poly1305}_{K_T}(\text{bin}_{128}(|L|)||T) \\
H_L &\leftarrow \text{Poly1305}_{K_L}(\text{NH}_{K_N}(\text{pad}_{128}(L)))
\end{align*}
\]

\[
\text{return } H_T \oplus H_L
\]

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 \(|L|/8192\)-bit input to Poly1305 with key \(K_L\) will differ. Since the sum of two \(\epsilon\)-\(\Delta\)U functions with independent keys is also \(\epsilon\)-\(\Delta\)U, if for all queries \(|T| \leq l_T\) and \(|L| \leq l_L\) then this composition is \(\epsilon\)-\(\Delta\)U, with:

\[
\epsilon = 2^{-128} + 2^{-103} \left[ \text{max}(128 + l_T, 256 [l_L/8192])/128 \right] = 2^{-128} + 2^{-103} \left[ \text{max}(1 + [l_T/128], 2 [l_L/8192]) \right]
\]

6.5 Usage limits

If we limit our Adiantum adversary to at most \(q\) queries each of which uses a tweak of length 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:

\[
\left(3(2^{-128}) + 2^{-103} \text{max}(1 + [l_T/128], 2 [l_M - 128]/8192])\right)^\left(q/2\right) + \text{Adv}^{\text{sc}}_{\text{KS}}(q + 1, 256 + 8832 + q(l_M - 128), t') + \text{Adv}^{\text{prp}}_{E_{\text{KE}}}(q, t')
\]

Assuming that the block and stream ciphers are strong, the advantage is dominated by the term for internal collisions: \(2^{-103} \text{max}(1 + [l_T/128], 2 [l_M - 128]/8192])^\left(q/2\right)\). 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 Poly1305 _K_ 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:

<table>
<thead>
<tr>
<th>Bytes of ciphertext</th>
<th>Advantage</th>
<th>Time on device (single-threaded)</th>
</tr>
</thead>
<tbody>
<tr>
<td>512GiB</td>
<td>2^{-47}</td>
<td>80 minutes</td>
</tr>
<tr>
<td>(2^{55})</td>
<td>2^{-15}</td>
<td>11 years</td>
</tr>
<tr>
<td>(2^{59})</td>
<td>0.8%</td>
<td>175 years</td>
</tr>
</tbody>
</table>

**Acknowledgements**

The authors would like to thank Yu Long Chen and our anonymous reviewers for helpful
comments, and for encouraging us to use the H-coefficient technique, which has led to a
much simpler proof.

**References**

[AB96] Ross Anderson and Eli Biham. “Two practical and provably secure block
ciphers: BEAR and LION”. In: *Fast Software Encryption: Third International
www.cl.cam.ac.uk/~rja14/Papers/bear-lion.pdf.

Internet-Draft draft-arciszewski-xchacha-02. IETF Secretariat, 2018-10. URL:

[Aum+08] Jean-Philippe Aumasson et al. “New Features of Latin Dances: Analysis of
Salsa, ChaCha, and Rumba”. In: *Fast Software Encryption*. Ed. by Kaisa
ISBN: 978-3-540-71039-4. DOI: 10.1007/978-3-540-71039-4_30. URL:

cipher”. Manuscript. 1999-02.

ia.cr/2013/404.

ia.cr/2015/585.

2017/560.


22 Adiantum: length-preserving encryption for entry-level processors


A 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.