The Calico Family of Authenticated Ciphers

Version 8

First round CAESAR Competition Submission Document

Designer/Submitter

Christopher Taylor, MSEE

mrcatid@gmail.com

March 15, 2014

Latest information available online:

http://www.calicoaead.com

1 Introduction

This is the official description of Calico, a first-round submission for the CAESAR competition. This document follows the structure guidelines from the CAESAR call for submissions (http://competitions.cr.yp.to/caesar-call.html).

Calico is a family of lightweight AEAD algorithms designed for simple and efficient network communications. It is motivated by realtime mobile and desktop applications (e.g.. online games, video/audio streaming, multi-user VR). Calico's flexibility, simplicity, and small software footprint allow it to be easily incorporated into existing systems.

Calico is based on the ChaCha-14 and ChaCha-20 [1], SipHash-2-4 [5], and BLAKE2 [6] algorithms, which were in no part designed or influenced by the submitter. The Calico combined construction was designed to enable features that are desirable for many applications, and these features are called out throughout this document to motivate the construction.

There are three parameter sets (also referred to as "modes") specified for Calico. The reference mode is mainly of interest in the CAESAR competition and corresponds to the reference code provided. Two additional practical modes are also fully specified: Stream and Datagram modes. These are individually optimized for common usage and are expected to be the main practical implementations of the Calico family.

2 Specification

This section provides a complete specification for Calico and its three parameter sets: Stream, Datagram, and Reference.

The Reference parameter set should be considered the **primary** set. The Datagram parameter set is the **secondary** set, and the Stream parameter set is **tertiary**. This ordering is motivated by the applicability of each set from general applications to specific ones. The reference parameters are the most general. The Datagram parameter set was designed with full-duplex communications between two parties in mind and is

applicable to applications using UDP sockets. And the Stream parameter set is also designed for full-duplex communications, but with the assumption that the messages are transported over a reliable and in-order channel such as TCP sockets.

This document is entirely self-contained and requires no external references to implement the described algorithms.

2.1 ChaCha Stream Cipher

The inputs to the ChaCha cipher function are a 32 byte key, an 8 byte public message number, and a number of rounds. Note that encryption and decryption proceeds identically.

2.1.1 Definitions

Word size := 32 bits for each value

$\oplus$:= Logical exclusive OR

+ := 32-bit integer addition

<<< := Logical left rotate

QUARTERROUND(a, b, c, d) function:

$a = a + b;\ d = d \oplus a;\ d = d <<< 16$

$c = c + d;\ b = b \oplus c;\ b = b <<< 12$

$a = a + b;\ d = d \oplus a;\ d = d <<< 8$

$c = c + d;\ b = b \oplus c;\ d = d <<< 16$

DOUBLEROUND(x0 .. x15) function:

$QUARTERROUND(x0, x4, x8, x12)$

$QUARTERROUND(x1, x5, x9, x13)$

$QUARTERROUND(x2, x6, x10, x14)$

$QUARTERROUND(x3, x7, x11, x15)$

$QUARTERROUND(x0, x5, x10, x15)$

$QUARTERROUND(x1, x6, x11, x12)$

$QUARTERROUND(x2, x7, x8, x13)$

$QUARTERROUND(x3, x4, x9, x14)$

2.1.2 Specification

During ciphering a state of 12 words is maintained. The lowest 32 bytes of the state are set in little-endian byte order to the 32 bytes of the key. The next 8 bytes are set to zero. And the remaining high 8 bytes are set to the 8 byte public message number in little-endian byte order.

For each block of 64 bytes of plaintext data, variables x0, x1, x2, x3 are set to the constants 0x61707865, 0x3320646e, 0x79622d32, 0x6b206574. And x4 through x15 are set to the 12 state words in the same order.

The DOUBLEROUND function is run for each set of two rounds to perform. Calico uses either 14 or 20 rounds. Then the original values of x0 … x15 are added to the final values produced by executions of DOUBLEROUND. The resulting values are considered the keystream.

The keystream is XORed into the plaintext to produce the ciphertext in little-endian byte order, making this a stream cipher:

$$\text{Ciphertext} = \text{Plaintext} \oplus \text{Keystream}$$

After each 64 bytes of plaintext, the value x9:x8 is taken as a 64-bit double-word and incremented by one, making the keystream different for each set of 64 bytes. Note that this double-word was initially set to zero and increments by one each time.

For the final block of plaintext, unused input words are zeroed, essentially padding the message with zeroes up to a multiple of 64 bytes.

2.2 SipHash-2-4 MAC

Calico specifies a slightly modified version of SipHash to support faster message authentication.

The inputs to Calico's SipHash-2-4 algorithm are a 16 byte Key, a message of variable length, and an 8 byte public message number.

2.2.1 Definitions

Word size := 64 bits for each value

$\oplus$:= Logical exclusive OR

+ := 64-bit integer addition

<<< := Logical left rotate

SIP_HALF_ROUND(a, b, c, d, s, t) function:

$a = a + b\ ;\ c = c + d$

$b = (b <<< s) \oplus a$

$d = (d <<< t) \oplus c$

$a = a <<< 32$

SIP_ROUND(v0, v1, v2, v3) function:

$SIP\_HALF\_ROUND(v0, v1, v2, v3, 13, 16)$

$SIP\_HALF\_ROUND(v2, v1, v0, v3, 17, 21)$

2.2.2 Specification

Two key 64-bit key words, K0 and K1, are constructed from the Key. K0 is set to the low 8 bytes in little-endian byte order. K1 is set to the high 8 bytes in little-endian byte order.

K0 is then XORed with the public message number in little-endian byte order, which is the only difference from normal SipHash-2-4.

Four values v0 .. v3 are constructed from K0, K1:

v0 =K0 $\oplus$ 0x736f6d6570736575

v1 =K1 $\oplus$ 0x646f72616e646f6d

v2 =K0 ⊕ 0x6c7967656e657261

v3 =K1 ⊕ 0x7465646279746573

For each set of 8 bytes of input data, denoted MW, treated as a 64-bit word in little-endian byte order the following algorithm executes.

v3 = v3 ⊕ MW

SIP_ROUND(v0, v1, v2, v3)

SIP_ROUND(v0, v1, v2, v3)

v0 = v0 ⊕ MW

For the final partial block, the high 8 bits are set to the low 8 bits of the length, denoted LAST, and the rest of the unused bytes are set to zero. The final rounds are performed on the padded buffer.

To produce the tag from the values v0 .. V3 a final mix is performed:

v2 = v2 ⊕ 0xFF

SIP_ROUND(v0, v1, v2, v3)

SIP_ROUND(v0, v1, v2, v3)

SIP_ROUND(v0, v1, v2, v3)

SIP_ROUND(v0, v1, v2, v3)

Tag = v0 ⊕ v1 ⊕ v2 ⊕ v3

Note that authentication of the MAC tag and generation of the MAC tag are the same operation.

2.3 BLAKE2b Hash Function

The BLAKE2b hash function has a lot of features that are mostly unused by Calico. BLAKE2b is only used for the Stream/Datagram modes for ratcheting the key. In this mode of operation, the input is a 128 byte block with all but the low 48 bytes set to zero and the output is a 64 byte hash of the input.

2.3.1 Definitions

Word size := 64 bits for each value

⊕ := Logical exclusive OR

+ := 64-bit integer addition

>>> := Logical right rotate

BLAKE2b CONSTANTS array:

0x6a09e667f3bcc908, 0xbb67ae8584caa73b, 0x3c6ef372fe94f82b,

0xa54ff53a5f1d36f1, 0x510e527fade682d1, 0x9b05688c2b3e6c1f,

0x1f83d9abfb41bd6b, 0x5be0cd19137e2179.

BLAKE2b uses round constants described by the following table:

```
SIGMA[12][16] = {
    {  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15 } ,
    { 14, 10,  4,  8,  9, 15, 13,  6,  1, 12,  0,  2, 11,  7,  5,  3 } ,
    { 11,  8, 12,  0,  5,  2, 15, 13, 10, 14,  3,  6,  7,  1,  9,  4 } ,
    {  7,  9,  3,  1, 13, 12, 11, 14,  2,  6,  5, 10,  4,  0, 15,  8 } ,
    {  9,  0,  5,  7,  2,  4, 10, 15, 14,  1, 11, 12,  6,  8,  3, 13 } ,
    {  2, 12,  6, 10,  0, 11,  8,  3,  4, 13,  7,  5, 15, 14,  1,  9 } ,
    { 12,  5,  1, 15, 14, 13,  4, 10,  0,  7,  6,  3,  9,  2,  8, 11 } ,
    { 13, 11,  7, 14, 12,  1,  3,  9,  5,  0, 15,  4,  8,  6,  2, 10 } ,
    {  6, 15, 14,  9, 11,  3,  0,  8, 12,  2, 13,  7,  1,  4, 10,  5 } ,
    { 10,  2,  8,  4,  7,  6,  1,  5, 15, 11,  9, 14,  3, 12, 13 , 0 } ,
    {  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15 } ,
    { 14, 10,  4,  8,  9, 15, 13,  6,  1, 12,  0,  2, 11,  7,  5,  3 }
};
```

G(r, i, a, b, c, d) function:

```
a = a + b + m[ SIGMA[r][2*i] ]
```

$d = (d \oplus a) >>> 32$

$c = c + d$

$b = (b \oplus c) >>> 24$

$a = a + b + m[\ SIGMA[r][2*i + 1]\ ]$

$d = (d \oplus a) >>> 16$

$c = c + d$

$b = (b \oplus c) >>> 63$

ROUND(r) function:

G(r, 0, v[0], v[4], v[8], v[12])

G(r, 1, v[1], v[5], v[9], v[13])

G(r, 2, v[2], v[6], v[10], v[14])

G(r, 3, v[3], v[7], v[11], v[15])

G(r, 4, v[0], v[5], v[10], v[15])

G(r, 5, v[1], v[6], v[11], v[12])

G(r, 6, v[2], v[7], v[8], v[13])

G(r, 7, v[3], v[4], v[9], v[14])

2.3.2 Specification

In this simplified mode of operation, the 48-byte message is padded up to the 128 byte block size with zeroes and loaded into 16 words m[0..15] used by the ROUND function. The mixing state v[0..15] is constructed by setting the low 8 words and the high 8 words to the CONSTANTS.

v[12] is effectively XORed by 128 (the counter increment). And v[14] is inverted, as this is the last (and only) block.

The BLAKE2b ROUND(r) function is then executed 12 times. The parameter r is initially 0 and increments by one each time.

The output hash is produced by XORing together the CONSTANTS with v[0..7] and v[8..15] such that the first hash word is v[0] ⊕ v[8] ⊕ 0x6a09e667f3bcc908. Each word is stored in little-endian byte order to yield a 64 byte hash.

Calico only uses the low 48 bytes of the hash for key ratcheting.

2.4 Reference Mode

This section describes the Reference Mode Calico algorithm, which is intended to be submitted as reference code for the CAESAR competition.

2.4.1 Reference Definitions

“Reference Mode” : Refers to the parameter set of Calico used for the CAESAR reference implementation.

“MAC Tag” : The authentication tag attached to each message, produced by SipHash-2-4.

“Cipher Key” : Keys the ChaCha-14 algorithm.

“MAC Key” : Keys the SipHash-2-4 algorithm.

2.4.2 Reference Parameters

In Reference mode:

	Size	*Notes*
Plaintext	0 .. $2^{63}-1$ bytes	*Authenticated, encrypted*
Associated Data	0 .. $2^{63}-1$ bytes	*Does not include the Public Message Number*
Secret Message Number	0 bytes	*Unused*
Public Message Number	8 bytes	*Ideally implicit*
MAC tag	8 bytes	*SipHash-2-4 tag*
MAC key	16 bytes	*SipHash-2-4 key*

Cipher key	32 bytes	*ChaCha14 key*

2.4.3 Reference Encryption

The reference encryption mode first executes the ChaCha-14 algorithm to produce the ciphertext from the 8 byte public message number and plaintext.

It will then XOR the 8 byte public message number into the low 8 little-endian bytes of the 16 byte MAC key, producing a unique 16 byte MAC key for this message.

The SipHash-2-4 algorithm is used to generate a tag for the ciphertext and associated data. The associated data is appended to the ciphertext for this algorithm. The result is an 8 byte MAC tag and the ciphertext.

2.4.4 Reference Decryption

If the public message number attached to the message has already been used or this cannot be determined, then the message will be rejected immediately. Typically a (roughly) 1024-bit sliding window is used to check for this with incrementing public message numbers.

It will then XOR the 8 byte public message number into the low 8 little-endian bytes of the 16 byte MAC key, producing a unique 16 byte MAC key for this message. The SipHash-2-4 algorithm is used to generate a tag for the ciphertext and associated data. The associated data is appended to the ciphertext for this algorithm. The result is an 8 byte MAC tag that is verified against the provided MAC tag. If this check fails, then the message is rejected.

Finally the ChaCha-14 algorithm is used to decrypt the ciphertext and reveal the original message plaintext.

2.5 Datagram Mode

This section describes the Datagram Mode Calico algorithm. This is not implemented in the reference version because it requires a different API. Instead, this mode serves as a practical

example of how the Calico construction results in a highly efficient AEAD algorithm.

2.5.1 Datagram Definitions

To ratchet keys, the BLAKE2b[6] hash algorithm is used. The input to the hash algorithm is the previous Cipher and MAC keys, concatenated explicitly so that the low 32 bytes correspond to the Cipher key and the high 16 bytes correspond to the MAC key. The output is truncated to 48 bytes, and the low 32 bytes are used as the new Cipher key, and the high 16 bytes are used as the new MAC key. This can be done in-place to erase the old Cipher and MAC keys, providing forward secrecy.

“Datagram Mode” : Refers to the parameter set of Calico designed for UDP-like data transport for full-duplex communication between two parties.

“Master Key” : The pre-shared secret key between all parties in the communication, either through online or offline agreement. It is used during initialization as input to the ChaCha-20 algorithm to expand into other keys.

“Ratchet bit” : A bit present in the overhead of each encrypted message indicating which key was used to encrypt the message.

“MAC Tag” : The authentication tag attached to each message, produced by SipHash-2-4.

“Cipher Key” : A key derived from the Master Key on initialization, used to key the ChaCha-14 algorithm.

“MAC Key” : A key derived from the Master Key on initialization, used to key the SipHash-2-4 algorithm.

“Role”: A parameter that describes the common role of a participant in the encrypted communications. As Calico supports either full-duplex or multicast communications, there are only two roles: Initiator or Responder. In multicast mode, the Responder is the data source and the Initiator is the data sink.

2.5.2 Datagram Parameters

The parameters listed here include, in addition to traditional variables such as allowed plaintext lengths, Calico-specific parameters that are used as inputs for the initialization, encryption, and decryption procedures.

In Datagram mode:

	Size	*Notes*
Plaintext	0 .. $2^{63}-1$ bytes	*Authenticated, encrypted*
Associated Data	0 .. $2^{63}-1$ bytes	*Does not include the Public Message Number*
Secret Message Number	0 bytes	*Unused*
Public Message Number	8 bytes	*Ideally implicit*
MAC tag	64 bits	*SipHash output*
Encryption Ratchet bit	1 bit	*Tracks rekeying*
Encryption MAC key	16 bytes	*Current key*
Encryption Cipher key	32 bytes	*Current key*
Decryption Ratchet bit	1 bit	*Tracks rekeying*
Decryption MAC key 0	16 bytes	*Current/next key*
Decryption MAC key 1	16 bytes	*Current/next key*
Decryption Cipher key 0	32 bytes	*Current/next key*
Decryption Cipher key 1	32 bytes	*Current/next key*
Master key	32 bytes	*ChaCha20 key*
Role	1 bit	*Initiator or responder role*
Last Ratchet Time	32 bits	*Timestamp in milliseconds since the last key ratchet*

2.5.3 Datagram Initialization

The input to initialization for each instance of Calico is explicitly the Role and Master Key. The Role (1 bit) in the communication is either Initiator (0) or Responder (1) and will be used to select keys appropriately. For multicast communication, the data source selects the Responder role, and the data sink selects the Initiator role.

The Master Key is used to key a ChaCha-20 instance, which produces 96 bytes of keystream. The Initiator uses the low 32 bytes as the Encryption Cipher key, and it uses the next 16 bytes as the Encryption MAC key, and it uses the next 32 bytes as the Decryption Cipher key 0, and it uses the high 16 bytes as the Decryption MAC key 0. The Responder uses the low 32 bytes as the Decryption Cipher key 0, and it uses the next 16 bytes as the Decryption MAC key 0, and it uses the next 32 bytes as the Encryption Cipher key, and it uses the high 16 bytes as the Encryption MAC key.

The key ratcheting algorithm, described below, is then run to ratchet the Decryption MAC/Cipher keys 0 and store the result in Decryption MAC/Cipher keys 1. The Decryption Ratchet bit is initialized to 0 to indicate that key 0 is active and key 1 is the next key. The Decryption Ratchet bit refers directly to one of these two keys, and after ratcheting occurs it will be updated to point to the new current key.

At this point it is recommended to set the outgoing and incoming public message number to zero, with the intent of incrementing it by one for each message.

2.5.4 Datagram Encryption

In a multicast setting only the Responder (the data source) is permitted to encrypt data, which can simplify implementation.

During encryption, a public message number is selected uniquely for the message.

It is recommended to increment the public message number by one each time, since this meets the goal with minimal complexity to guarantee that public message numbers are not reused. In this case, simply checking if the high bit of the next 64-bit public message number

is set is enough to determine if encryption can proceed or not. If the high bit is set than there are no unique numbers left to use. An additional benefit of this incrementing approach is that the Public Message Number does not ever need to be explicitly transmitted. It can be implicitly understood from the order in which messages are received in Stream mode.

At this point if the Role is Initiator, the current time and the Last Ratchet Time are compared, minding register overflows. If more than 2 minutes has gone by and the Encryption Ratchet Bit matches the Decryption Ratchet Bit, indicating that the Responder is synchronized with the current key, then the Encryption Cipher key is ratcheted. After ratcheting the key as described in the definitions section, the Encryption Ratchet bit is flipped to indicate that the ratchet occurred.

The 32 byte Encryption Cipher key is used as input to the ChaCha-14 algorithm along with the Plaintext data and the 8 byte Public Message Number. The result is Ciphertext of the same length as the Plaintext data.

The SipHash-2-4 algorithm is slightly modified in Calico to support associated data. Each message tag is generated using a unique Message MAC key, which is constructed by XORing the Public Message Number in little-endian byte order into the low half of the Encryption MAC key. This guarantees that each Message MAC key is unique for each message. This implicitly authenticates the Public Message Number, as the same Ciphertext sent with two message will be overwhelmingly unlikely to have the same tag. This is efficient because it requires only a single XOR operation to implement.

The Message MAC key is used as input to the SipHash-2-4 algorithm along with the Ciphertext data, concatenated by the Associated Data such that the Associated Data comes second. The output is the 8 byte message MAC tag.

In Datagram mode, it is recommended to truncate the Public Message Number to 23 bits (losing its high bit) so that the Encryption Ratchet Bit can be stored in the low bit. Note that since the Public Message Number is ideally incremented by one each time, it is trivial to write

branchless code to reconstruct the full implicit number from its low bits based on the last number received. This trick saves 5 bytes of overhead on each message to encrypt.

The output of the Datagram encryption process is: Message Ciphertext, Message Associated Data, Message MAC tag, Message Ratchet Bit, and Public Message Number. The Public Message Number is ideally truncated to 23 bits as recommended above. As a result the extra data added to each encrypted message is just 11 bytes.

2.5.5 Datagram Decryption

Stream mode decryption involves first authenticating the input message received.

If the public message number attached to the message has already been used or this cannot be determined, then the message will be rejected immediately. Typically a (roughly) 1024-bit sliding window is used to check for this with incrementing public message numbers.

If the received Message Ratchet Bit differs from the Decryption Ratchet Bit, then the ratcheted key is used in place of the current Decryption MAC/Cipher key during decryption. In practice this means that the ratchet bit selects which key 0 or 1 to use. When ratcheting happens the bit flips, and the inactive key is the ratcheted version of the key specified by the ratchet bit.

The expected Message MAC tag is constructed from the Decryption MAC key as in encryption by XORing the low 8 bytes of the Decryption MAC key by the Public Message Number in little-endian byte order. The Ciphertext is appended by the Associated Data and then used as input to the SipHash-2-4 to produce an 8 byte tag, which is then truncated as before to produce the 63-bit truncated expected MAC tag. This is checked in constant time to verify that the expected tag matches the 63-bit received tag. If these tags do not match then decryption fails out immediately to avoid allowing unauthenticated data from affecting the rest of the state.

The Decryption Ratchet Key is set to the Message Ratchet key after authentication. On the Responder, if the Decryption Ratchet Key was flipped then the Responder will ratchet its key

and flip its Encryption Ratchet Key, which also serves to acknowledge the Initiator's ratchet. A 1 minute timer is started on the Responder side to ratchet the Decryption Ratchet Key to erase the Initiator's encryption keys, which gives plenty of time in most implementations for out-of-order messages from the Initiator to finish arriving, avoiding data loss during rekeying.

At this point it is recommended to update the Public Message Number if it is incrementing by one each time, after authentication.

The Decryption Cipher key is then used as input to the ChaCha-14 algorithm along with the Ciphertext and Public Message Number. The output should be the original Plaintext that was encrypted.

2.6 Stream Mode

This section describes the Stream Mode Calico algorithm. This is not implemented in the reference version because it requires a different API. Instead, this mode serves as a practical example of how the Calico construction results in a highly efficient AEAD algorithm.

2.6.1 Stream Definitions

"Stream Mode" : Refers to the parameter set of Calico designed for TCP-like data transport.

"Master Key" : The pre-shared secret key between all parties in the communication, either through online or offline agreement. It is used during initialization as input to the ChaCha-20 algorithm to expand into other keys.

"Ratchet bit" : A bit present in the overhead of each encrypted message indicating which key was used to encrypt the message.

"MAC Tag" : The authentication tag attached to each message, produced by SipHash-2-4.

"Cipher Key" : A key derived from the Master Key on initialization, used to key the ChaCha-14 algorithm.

"MAC Key" : A key derived from the Master Key on initialization, used to key the SipHash-2-4 algorithm.

"Role": A parameter that describes the common role of a participant in the encrypted communications. As Calico supports either full-duplex or multicast communications, there are only two roles: Initiator or Responder. In multicast mode, the Responder is the data source and the Initiator is the data sink.

2.6.2 Stream Parameters

The parameters listed here include, in addition to traditional variables such as allowed plaintext lengths, Calico-specific parameters that are used as inputs for the initialization, encryption, and decryption procedures.

The accepted parameter ranges for Stream and Datagram modes are mostly identical, excepting that Stream mode MAC tags are one bit shorter. This matches the security goals of Calico.

In Stream mode:

	Size	*Notes*
Plaintext	$0 .. 2^{63} - 1$ bytes	*Authenticated, encrypted*
Associated Data	$0 .. 2^{63} - 1$ bytes	*Does not include the Public Message Number*
Secret Message Number	0 bytes	*Unused*
Public Message Number	8 bytes	*Ideally implicit*
Message MAC tag	63 bits	*Carved by ratchet bit*
Message MAC key	16 bytes	*Unique per-message key*
Encryption Ratchet bit	1 bit	*Triggers rekeying*
Encryption MAC key	16 bytes	*SipHash-2-4 key*
Encryption Cipher key	32 bytes	*ChaCha14 key*
Decryption Ratchet bit	1 bit	*Tracks rekeying*
Decryption MAC key	16 bytes	*SipHash-2-4 key*
Decryption Cipher key	32 bytes	*ChaCha14 key*

Master key	32 bytes	*ChaCha20 key*
Role	1 bit	*Initiator or responder role*
Last Ratchet Time	32 bits	*Timestamp in milliseconds since the last key ratchet*

2.6.3 Stream Initialization

The input to initialization for each instance of Calico is explicitly the Role and Master Key. The Role (1 bit) in the communication is either Initiator (0) or Responder (1) and will be used to select keys appropriately. For multicast communication, the data source selects the Responder role, and the data sink selects the Initiator role.

The Master Key is used to key a ChaCha-20 instance, which produces 96 bytes of keystream. The Initiator uses the low 32 bytes as the Encryption Cipher key, and it uses the next 16 bytes as the Encryption MAC key, and it uses the next 32 bytes as the Decryption Cipher key, and it uses the high 16 bytes as the Decryption MAC key. The Responder uses the low 32 bytes as the Decryption Cipher key, and it uses the next 16 bytes as the Decryption MAC key, and it uses the next 32 bytes as the Encryption Cipher key, and it uses the high 16 bytes as the Encryption MAC key.

Each instance of Calico initializes the Last Ratchet Time to a current timestamp.

At this point it is recommended to set the outgoing and incoming public message number to zero, with the intent of incrementing it by one for each message.

2.6.4 Stream Encryption

In a multicast setting only the Responder (the data source) is permitted to encrypt data, which can simplify implementation.

During encryption, a public message number is selected uniquely for the message.

It is recommended to increment the public message number by one each time, since this

meets the goal with minimal complexity to guarantee that public message numbers are not reused. In this case, simply checking if the high bit of the next 64-bit public message number is set is enough to determine if encryption can proceed or not. If the high bit is set than there are no unique numbers left to use. An additional benefit of this incrementing approach is that the Public Message Number does not ever need to be explicitly transmitted. It can be implicitly understood from the order in which messages are received in Stream mode.

At this point if the Role is Initiator, the current time and the Last Ratchet Time are compared, minding register overflows. If more than 2 minutes has gone by and the Encryption Ratchet Bit matches the Decryption Ratchet Bit, indicating that the Responder is synchronized with the current key, then the Encryption Cipher key is ratcheted. After ratcheting the key as described below, the Encryption Ratchet bit is flipped to indicate that the ratchet occurred.

To ratchet keys, the BLAKE2b[6] hash algorithm is used. The input to the hash algorithm is the previous Cipher and MAC keys, concatenated explicitly so that the low 32 bytes correspond to the Cipher key and the high 16 bytes correspond to the MAC key. The output is truncated to 48 bytes, and the low 32 bytes are used as the new Cipher key, and the high 16 bytes are used as the new MAC key. This can be done in-place to erase the old Cipher and MAC keys, providing forward secrecy.

The 32 byte Encryption Cipher key is used as input to the ChaCha-14 algorithm along with the Plaintext data and the 8 byte Public Message Number. The result is Ciphertext of the same length as the Plaintext data.

The SipHash-2-4 algorithm is slightly modified in Calico to support associated data. Each message tag is generated using a unique Message MAC key, which is constructed by XORing the Public Message Number in little-endian byte order into the low half of the Encryption MAC key. This guarantees that each Message MAC key is unique for each message. This implicitly authenticates the Public Message Number, as the same Ciphertext sent with two message will be overwhelmingly unlikely to have the same tag. This is efficient because it

requires only a single XOR operation to implement.

The Message MAC key is used as input to the SipHash-2-4 algorithm along with the Ciphertext data, concatenated by the Associated Data such that the Associated Data comes second. The output is the 8 byte message MAC tag.

In Stream mode, the MAC tag is truncated to 63 bits (losing its high bit) so that the Encryption Ratchet Bit can be stored in the low bit.

The output of the Stream encryption process is: Message Ciphertext, Message Associated Data, Message MAC tag, and Message Ratchet Bit. The Public Message Number is ideally implicit. As a result the extra data added to each encrypted message is just 8 bytes.

2.6.5 Stream Decryption

Stream mode decryption involves first authenticating the input message received.

If the received Message Ratchet Bit differs from the Decryption Ratchet Bit, then the ratcheted key is used in place of the current Decryption MAC/Cipher key during decryption.

Note that it is recommended but not necessary to pre-calculate the next ratcheted key as is done in Datagram mode to avoid allowing an attacker from being able to trigger a full BLAKE2b calculation in the decryption code.

The expected Message MAC tag is constructed from the Decryption MAC key as in encryption by XORing the low 8 bytes of the Decryption MAC key by the Public Message Number in little-endian byte order. The Ciphertext is appended by the Associated Data and then used as input to the SipHash-2-4 to produce an 8 byte tag, which is then truncated as before to produce the 63-bit truncated expected MAC tag. This is checked in constant time to verify that the expected tag matches the 63-bit received tag. If these tags do not match then decryption fails out immediately to avoid allowing unauthenticated data from affecting the rest of the state.

The Decryption Ratchet Key is set to the Message Ratchet key after authentication. On the

Responder, if the Decryption Ratchet Key was flipped then the Responder will ratchet its key and flip its Encryption Ratchet Key, which also serves to acknowledge the Initiator's ratchet. Since message cannot arrive out of order, the Responder will also immediately ratchet the key for the Initiator, erasing both local and remote copies of that key.

At this point it is recommended to update the Public Message Number if it is incrementing by one each time, after authentication.

The Decryption Cipher key is then used as input to the ChaCha-14 algorithm along with the Ciphertext and Public Message Number. The output should be the original Plaintext that was encrypted.

3 Security Goals

3.1 Bits of Security

In terms of bits of security, the following security levels are claimed for attacks with 50% probability of success.

	Bits of security
Confidentiality for the plaintext	127
Integrity for the plaintext	63*
Integrity for the associated data	63*
Integrity for the public message number	63*
Confidentiality for erased, old keys in Stream/Datagram modes	127

* Note that Calico Stream mode reduces this to 62 bits of security.

3.2 Nonce Reuse

Users are required to use the concatenation of the secret message number and the public message number as a nonce, i.e., that the cipher may lose all integrity and confidentiality if the legitimate key holder uses the same (secret message number,public message number) pair to encrypt two different (plaintext,associated data) pairs under the same key.

With different keys, using the a public message number from an earlier key does not lead to a compromise. This assumes that the key is chosen uniformly at random and that it is not linearly related to the public message number.

4 Security Analysis

4.1 Security of Primitives

Calico leverages existing security analysis of the primitives used.

The ChaCha cipher has gone through a lot of analysis since it was proposed[1][2][3][4]. In [4], a proof for security against differential cryptanalysis was presented up to 15 rounds. As a result Calico uses the 14-round variant.

The SipHash MAC is fairly new but is based on the ChaCha round function and is simple in design. Its security analysis is provided in [5].

The BLAKE2 hash function is also fairly new and is similarly based on the ChaCha round function. Its security analysis is provided in [6].

4.2 Explicit Limitations

Calico Stream/Datagram modes are designed for network communications. These modes are not, for example, designed with virtual file system encryption in mind. Generalizing to cover all possible use cases of encryption may lead to overly complex and inefficient software.

The forward secrecy of Calico is only guaranteed for two-way communications between two endpoints. For e.g. Multicast UDP applications, synchronizing rekeying is the responsibility of the application. Multicast rekeying is not covered by this algorithm because it would unnecessarily add complexity, as multicast rekeying can be safely implemented outside of the Calico algorithm.

Calico does not hide the length of the original encrypted data. This allows the application developer to decide on a trade-off between security and encrypted data size.

Calico encrypts/decrypts plaintext message data from zero to $2^{63}-1$ bytes in length. It authenticates associated data from zero to $2^{63}-1$ bytes in length, not including the public message number. It refuses to encrypt more than $2^{63}-1$ messages. These practical limitations simplify input checking and avoid possible platform-specific software bugs related to sign bits.

4.3 Nonce Re-Use

Calico specifies that Public Message Numbers should be unique for each key, which prevents attacks caused by nonce re-use.

In the Stream/Datagram modes the IV is XORed into the MAC key, which does not affect the likelihood for MAC key collision. This is because the Public Message Numbers are unique, so for a given fixed key all of the resulting keys are unique.

4.4 Encrypt-then-MAC

Calico uses the well-known Encrypt-then-MAC construction to allow the SipHash MAC tag to prevent the cipher from being affected by unauthenticated ciphertext.

4.5 Constant-Time Execution

All of the Calico algorithms run in constant-time without key-dependent memory accesses, and do not present timing or power analysis (PA) side channels.

5 Features

The design of Calico is somewhat conservative, reusing existing primitives. The most innovative features of the Calico algorithm are found in how the primitives are used. Forward secrecy is made part of the algorithm design by forcing periodic rekeying in a way that does not lose data. This enables use Calico to be a completely separate choice from the method of key agreement. As a result it is more flexible for incorporation into existing codebases.

Calico also fits the algorithm to the way that the data is transported. Applications that currently use TCP or some other reliable-in-order transport will use the stream-oriented mode, called “Stream mode”. And other applications that currently use UDP or some other unreliable/unordered transport will use a datagram-oriented mode, called “Datagram mode”. By optimizing for each of these use cases (Stream and Datagram), lower memory usage and message size are achieved in each case. Calico does not require a specific packet format, allowing it to be fit to existing applications that have a legacy format.

Over AES-GCM, users of Calico will benefit from much lower packet overhead and much higher throughput on mobile devices. Furthermore the Calico software is much simpler to implement and easier to audit.

6 Design Rationale

6.1 Backdoors

It may be possible to hide backdoors in the constants used to initialize each of the ARX algorithms used by Calico. However these constants are well-explained by the designers[1] [5][6] and are thus unlikely to be maliciously designed.

The designer has not hidden any weaknesses in this cipher. It is unlikely that the primitives it is based on have backdoors, and the submitter is not able to speak on behalf of the designers of the primitives used, but it is understood that no backdoor was installed in the design of any part of Calico.

6.2 Parameters

The parameters for Stream and Datagram modes are fundamentally identical to reduce code complexity.

The accepted parameter ranges for Stream and Datagram modes are mostly identical, excepting that Stream mode MAC tags are one bit shorter, allowing software implementations to save one byte of overhead.

Note that the datagram mode, when used for multicast, is unnecessarily transmitting a Ratchet bit to all the recipients. This is an acceptable tradeoff for simplicity because in practice, the Ratchet bit can be carved out of the truncated public message number, which does not need more than 20 or so bits, so there is no extra overhead for the Ratchet bit in practice even in a multicast setting.

6.3 Encrypt-then-MAC

Calico uses the well-known Encrypt-then-MAC construction. This allows invalid packets to be rejected by a receiver more quickly because the decryption algorithm does not need to be run.

6.4 SipHash-2-4 Tweak

Calico tweaks the SipHash-2-4 function to incorporate the public message number. This is an optimization that improves the speed of the SipHash function.

7 Intellectual Property and Consent

7.1 Intellectual Property

The submitter hereby declares that Calico will remain patent-free and open-source.

Calico is built on existing, patent-free primitives:

1. ChaCha14, ChaCha20 by Daniel J. Bernstein [1]. The submitter is unaware of any patent claims made by the designer.

2. SipHash-2-4 by Jean-Philippe Aumasson and Daniel J. Bernstein [5]. The submitter is unaware of any patent claims made by the designers.

3. BLAKE2 by Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. Patent-free status is declared explicitly in [6] by the designers.

If any of this information changes, the submitter will promptly (and within at most one month) announce these changes on the crypto-competitions mailing list.

7.2 Consent to CAESAR Committee

The submitter hereby consents to all decisions of the CAESAR selection committee regarding the selection or non-selection of this submission as a second-round candidate, a third-round candidate, a finalist, a member of the final portfolio, or any other designation provided by the committee.

The submitter understands that the committee will not comment on the algorithms, except that for each selected algorithm the committee will simply cite the previously published analyses that led to the selection of the algorithm.

The submitter understands that the selection of some algorithms is not a negative comment regarding other algorithms, and that an excellent algorithm might fail to be selected simply because not enough analysis was available at the time of the committee decision. The submitter acknowledges that the committee decisions reflect the collective expert judgements of the committee members and are not subject to appeal. The submitter understands that if they disagree with published analyses then they are expected to promptly and publicly respond to those analyses, not to wait for subsequent committee decisions.

The submitter understands that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.

Chris Taylor, MSEE

Mountain View, CA, USA

March 15, 2014

Bibliography

[1] Daniel J. Bernstein “ChaCha, a variant of Salsa20”, University of Illinois at Chicago, Jan 2008. http://cr.yp.to/chacha/chacha-20080128.pdf

[2] Jean-Philippe Aumasson “New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba”, 2008. https://eprint.iacr.org/2007/472.pdf

[3] Tsukasa Ishiguro “Latin Dances Revisited: New Analytic Results of Salsa20 and ChaCha”, 2012. https://eprint.iacr.org/2012/065.pdf

[4] Nicky Mouha and Bart Preneel “Towards Finding Optimal Differential Characteristics for ARX”, 2013. http://eprint.iacr.org/2013/328.pdf

[5] Jean-Philippe Aumasson and Daniel J. Bernstein “SipHash: a fast short-input PRF”, 2013. https://131002.net/siphash/siphash.pdf

[6] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O’Hearn, Christian Winnerlein “BLAKE2: simpler, smaller, fast as MD5”, Jan 2013. https://blake2.net/blake2_20130129.pdf