LIONESS Wide-Block-Cipher

Yawning Angel

Table of Contents

Abstract

This document defines the LIONESS Wide-Block-Cipher construct, and provides a concrete parameterization based around the BLAKE2b hash algorithm and ChaCha20 stream cipher.

This document does not introduce any new crypto, but is meant to serve as a stable reference and implementation guide.

1. Introduction

LIONESS is a provably secure wide-block-cipher proposed by Anderson and Biham in [AB96]. Internally it is a four round unbalanced Feistel cipher comprised of a keyed hash function and a stream cipher. It supports processing large abitrary sized blocks, with a minimum block size imposed by the choice of primitives, and has the property such that each bit of the ciphertext is dependent on every single bit of the plaintext and vice versa.

1.1 Conventions Used in This Document

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT",”SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC2119.

- `x | y` is the concatenation of x and y
- `x ^ y` is the bitwise XOR of x and y
- A `byte` is an 8-bit octet

`x[a:b]` is the sub-vector of x where a/b denote the start/end byte indexes (inclusive-exclusive). a/b may be omitted to signify the start/end of the vector x respectively.

2. The LIONESS Wide-Block-Cipher Construct

LIONESS is parameteriezed with a keyed hash function (MAC) and a stream cipher as follows.

`H(k, m)` is a keyed hash function (MAC) with the key `k`, for a message `m` of arbitrary size. Let `Hash-Length` denote the size of the output and the key in bytes.

`H()` MUST be collision resistant or preimage resistant, and implementations SHOULD pick algorithms that provide both properties.

`S(k)` is a pseudo random function (stream cipher) which given the `HashLength` sized input `k` will generate an output of arbitrary size.

`S()` MUST be key recovery resistant.

2.1 LIONESS Encryption

```
LIONESS-Encrypt(k1, k2, k3, k4, plaintext) -> ciphertext
```

Inputs:

`k1,k2,k3,k4` Round keys, each HashLength bytes in size.

`plaintext` The plaintext to encrypt, greater than `HashLength` bytes in size.

Output:

`ciphertext` The resulting ciphertext.

The output of LIONESS-Encrypt is calculated as follows:

```
L = plaintext[0:HashLength]
R = plaintext[HashLength:]
R = R ^ S(L ^ k1)
L = L ^ H(k2, R)
R = R ^ S(L ^ k3)
L = L ^ H(k4, R)
ciphertext = L | R
```

2.2 LIONESS Decryption

```
LIONESS-Decrypt(k1, k2, k3, k4, ciphertext) -> plaintext
```

Inputs:

`k1,k2,k3,k4` Round keys, each `HashLength` bytes in size.

`ciphertext` The ciphertext to decrypt, greater than `HashLength` bytes in size.

Output:

`plaintext` The resulting plaintext.

The output of LIONESS-Decrypt is calculated as follows:

```
L = ciphertext[0:HashLength]
R = ciphertext[HashLength:]
L = L ^ H(k4, R)
R = R ^ S(L ^ k3)
L = L ^ H(k2, R)
R = R ^ S(L ^ k1)
plaintext = L | R
```

3. LIONESS-BLAKE2b-ChaCha20

LIONESS-BLAKE2b-ChaCha20 is a concrete parameterization of LIONESS based around the BLAKE2b RFC7693 hash algorithm and ChaCha20 RFC7539 stream cipher. It provides a security level of at least 256 bits, and supports a per-call initialization vector.

Plaintext and Ciphertext MUST NOT exceed `32 + ((1 \<\< 32) \* 64)` bytes.

For sections 3.1 and 3.2:

Let `BLAKE2b(k, m)` return the BLAKE2b digest calculated with key `k`, and message `m`, truncated to 32 bytes.

Let `ChaCha20(k, n, m)` return the ChaCha20 encrypted ciphertext with key `k`, nonce `n`, and message `m`, with the counter initialized to `0`.

3.1 LIONESS-BLAKE2b-ChaCha20 Encryption

`LIONESS-BLAKE2b-ChaCha20-Encrypt(key, iv, plaintext) -> ciphertext`

Inputs:

`key` The key, 128 bytes in size. `iv` The initialization vector, 48 bytes in size. `plaintext` The plaintext to encrypt, greater than 32 bytes in size.

Output:

`ciphertext` The resulting ciphertext.

The output of LIONESS-BLAKE2b-ChaCha20-Encrypt is calculated as follows:

```
k1 = key[0:32]
k2 = key[32:64]
k3 = key[64:96]
k4 = key[96:128]
iv1 = iv[0:12]
iv2 = iv[12:24]
iv3 = iv[24:36]
iv4 = iv[36:48]

L = ciphertext[0:32]
R = ciphertext[32:]
```

```
R = ChaCha20(L ^ k1, iv1, R)
L = L ^ BLAKE2b(k2 | iv2, R)
R = ChaCha20(L ^ k3, iv3, R)
L = L ^ BLAKE2b(k4 | iv4, R)
ciphertext = L | R
```

3.2 LIONESS-BLAKE2b-ChaCha20 Decryption

`LIONESS-BLAKE2b-ChaCha20-Decrypt(key, iv, ciphertext) -> plaintext`

Inputs:

`key` The key, 128 bytes in size.

`iv` The initialization vector, 48 bytes in size.

`ciphertext` The ciphertext to decrypt, greater than 32 bytes in size.

Output:

`plaintext` The resulting plaintext.

The output of LIONESS-BLAKE2b-ChaCha20-Decrypt is calculated as follows:

```
k1 = key[0:32]
k2 = key[32:64]
k3 = key[64:96]
k4 = key[96:128]
iv1 = iv[0:12]
iv2 = iv[12:24]
iv3 = iv[24:36]
iv4 = iv[36:48]

L = ciphertext[0:32]
R = ciphertext[32:]
L = L ^ BLAKE2b(k4 | iv4, R)
R = ChaCha20(L ^ k3, iv3, R)
L = L ^ BLAKE2b(k2 | iv2, R)
R = ChaCha20(L ^ k1, iv1, R)
plaintext = L | R
```

4. Implementation Considerations

When choosing the underlying stream cipher or MAC, implementors may wish to consider the initialization overhead such as key scheduling, as the performance impact can be non-negligible depending on algorithm choice.

5. Security Considerations

When parameterizing the LIONESS construct care MUST be taken to pick cryptographic primitives that meet the requirements specified in Section 2.1. Depending on the primitive chosen for `S()`, there may be a maximum block size imposed by the maximum amount of data that `S()` may encrypt with a given key.

Care MUST be taken to avoid leaking sensitive information via side-channels, however this is primarily influenced by the algorithms and implementations selected for `H()` and `S()` than the LIONESS construct itself.

No claims are made regarding the security of LIONESS when the same key material is used to encrypt multiple blocks, beyond those made in MPRA11. Conservative users may wish to avoid this behavior, use LIONESS as the building block for standard block cipher constructs that take initialization vectors, or incorporate initialization vectors in the `H()` and `S()` calls.

Appendix A. References

Appendix A.1 Normative References

Appendix A.2 Informative References

Appendix B. LIONESS-ChaCha20-BLAKE2b Test Vector

Appendix C. Citing This Document

Appendix C.1 Bibtex Entry

Note that the following bibtex entry is in the IEEEtran bibtex style as described in a document called “How to Use the IEEEtran BIBTEX Style”.

```
@online{LionessSpec,
title = {The LIONESS Wide-Block-Cipher Specification},
author = {Yawning Angel},
url = {https://github.com/katzenpost/katzenpost/blob/main/docs/specs/lioness.md},
year = {2017}
}
```

AB96

```
Anderson, R., Biham, E.,
"Two Practical and Provably Secure Block Ciphers: BEAR and LION",
1996
```

MPRA11

```
Maines, L., Piva, M., Rimoldi, A., Sala, M.,
"On the provable security of BEAR and LION schemes",
arXiv:1105.0259,
May 2011,
https://arxiv.org/abs/1105.0259
```

RFC2119

Bradner, S.,
"Key words for use in RFCs to Indicate Requirement Levels",
BCP 14, RFC 2119, DOI 10.17487/RFC2119,
March 1997,
http://www.rfc-editor.org/info/rfc2119

RFC7539

Nir, Y. and A. Langley,
"ChaCha20 and Poly1305 for IETF Protocols",
RFC 7539, DOI 10.17487/RFC7539,
May 2015,
http://www.rfc-editor.org/info/rfc7539

RFC7693

Saarinen, M-J., Ed., and J-P. Aumasson,
"The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)",
RFC 7693, DOI 10.17487/RFC7693,
November 2015,
http://www.rfc-editor.org/info/rfc7693