BLAKE2X

2016.12.03

https://blake2.net

Jean-Philippe Aumasson (@veorq)
Samuel Neves (@sevenps)
Zooko Wilcox-O'Hearn (@zooko)
Christian Winnerlein (@codesinchaos)

Abstract. We present BLAKE2X, a new family of BLAKE2 hash functions that produce hash values of arbitrary length—that is, extensible-output functions (XOFs). You can use BLAKE2X within signatures schemes such as EdDSA or RSA-FDH, and also as a DRBG or KDF.

1 Introduction

Sometimes a 512-bit hash is not enough. For example, Ed521 signatures [3] need a 1056-bit hash; RSA-FDH [2] needs a 4096-bit hash when using a 4096-bit modulus. Since BLAKE2 was limited to 512-bit hashes, and based on user demand, we present BLAKE2X, a family of extensible-output functions (XOFs) based on BLAKE2 instances. BLAKE2X can securely produce hash values of arbitrary size.

We designed BLAKE2X to minimize the changes from an existing BLAKE2 implementation. BLAKE2X only needs to change the value of certain parameters in BLAKE2's parameter block, and the implementation of a simple counter-like logic.

As a by-product, BLAKE2X provides two additional functionalities:

- Deterministic random bit generator (DRBG): Given a high-entropy seed, BLAKE2X produces a stream of up to 256 GiB.
- Key derivation function (KDF): Given input key material (and an optional salt, as allowed by the BLAKE2 parameter block), BLAKE2X computes a key of up to $2^{32} - 2$ bytes (about 4 GiB). BLAKE2X's extract-then-expand scheme is similar (but not identical) to that of HKDF [5, 6].

2 Specification of BLAKE2X

A BLAKE2X instance can be derived from any BLAKE2 instance, be it BLAKE2b, BLAKE2s, the parallel versions BLAKE2bp and BLAKE2sp, or any custom version defined as per the BLAKE2 specifications.

BLAKE2X requires a slightly modified version of the parameter block, where the length of the "Node offset" field is reduced from 8 to 4 bytes for 64-bit BLAKE2, and from 6 to 4 bytes for 32-bit BLAKE2. This change does not break compatibility with already used BLAKE2 versions. Tables 1 and 2 show the new parameter with the extra field "XOF digest length".

Given a 64-bit BLAKE2 instance such as BLAKE2b, BLAKE2X works as follows to compute an ℓ-byte hash of some message M:

Offset	0	1	2	3
0	Digest length	Key length	Fanout	Depth
4	Leaf length			
8	Node offset			
12	XOF digest length			
16	Node depth	Inner length	RFU	
20 **24** **28**	RFU			
32 ... **44**	Salt			
48 ... **60**	Personalization			

Table 1: BLAKE2b parameter block structure (offsets in bytes).

Offset	0	1	2	3
0	Digest length	Key length	Fanout	Depth
4	Leaf length			
8	Node offset			
12	XOF digest length		Node depth	Inner length
16 **20**	Salt			
24 **28**	Personalization			

Table 2: BLAKE2s parameter block structure (offsets in bytes).

1. Set the "XOF digest length" parameter to a little-endian encoding of ℓ. If the output length is *not known* in advance, ℓ should be set to the maximal value $2^{32} - 1$ (a feature similar to that of KMAC [8, §4.3.1]).

2. Compute the hash of M using the underlying BLAKE2 instance as usual. Call the result H_0. When a tree mode is used (such as BLAKE2bp), this corresponds to the root of the tree.

3. Create the hash function instance B2 from the BLAKE2 instance used, where:

   - "Key length" is set to 0 (even if the root hash was keyed)
   - "Fanout" is set to 0 (unlimited)
   - "Maximal depth" is set to 0
   - "Leaf maximal byte length" is set to 32 for BLAKE2Xs, and 64 for BLAKE2Xb
   - "XOF digest length" is set to ℓ
   - "Node depth" is set to 0 (leaves)
   - "Inner hash byte length" is set to 32 for BLAKE2Xs and 64 for BLAKE2Xb
   - Other fields are left to the same values as in the underlying BLAKE2 instance
   - $\mathrm{B2}(\mathrm{i}, \mathrm{j}, \mathrm{X})$ denotes the hash of X using i as node offset and j as digest length

4. The final hash is computed as follows:

$$\mathsf{B2}(0, 64, \mathrm{H}_0) \| \mathsf{B2}(1, 64, \mathrm{H}_0) \| \mathsf{B2}(2, 64, \mathrm{H}_0) \| \cdots \| \mathsf{B2}(\lfloor \ell/64 \rfloor, \ell \bmod 64, \mathrm{H}_0)$$

With a 32-bit BLAKE2 (such as BLAKE2s), 64 is replaced with 32 in the last expression, and ℓ is set to $2^{16} - 1$ when the output length is unknown in advance.

You can see BLAKE2X as the computation of an inverted tree on top of a BLAKE2 instance: first hash the message, then expands it to enough bytes to form the hash. This construction is similar to that informally described in [1, §7].

When a key is used, the key is only processed by the BLAKE2 instance that computes H_0. The B2 instances are never keyed.

We call BLAKE2Xbℓ the BLAKE2X version based on BLAKE2b and producing ℓ-byte hashes. For example, BLAKE2Xb132 produces 1056-bit digests. By default the length is in bytes but you may for example write BLAKE2Xb16MiB instead of BLAKE2Xb16777216.

When the number of bytes to generate is known in advance, ℓ must be set to that value. In this case, ℓ is at most $2^{32} - 2$ (4 gibibytes minus two bytes) on BLAKE2Xb, and at most $2^{16} - 2$ (64 kibibytes minus two bytes) on BLAKE2Xs. If BLAKE2X is used as a DRBG, it must therefore be reseeded before reaching this maximum output (and preferably before that for better backward secrecy.)

When the number of bytes to generate is *not known* in advance, 64-bit BLAKE2X can generate up to 2^{32} blocks of 64 bytes, that is, 256 GiB; 32-bit BLAKE2X can generate up to 2^{32} blocks of 32 bytes, that is, 128 GiB.

Note that an ℓ smaller or equal than 64 is acceptable, there's just no reason to use it since BLAKE2b already provides hashes of 64 bytes or less.

3 Security

The BLAKE2X construction is similar to the counter construction proven secure in [4, §5]. The underlying BLAKE2 instances are proven to be secure [7]—indifferentiable from a random oracle—and the parameter block values ensure a domain separation between the compressing and the expanding hash trees. The resulting hash is thus expected to be at least as secure as the underlying hash, and in particular to provide $\mathrm{n}/2$-bit collision resistance, where n is the length of H_0 (or the inner hash length, if this is smaller than the digest length). For the highest security level, the digest length n should be set to its maximal value, that is, to 32 bytes for 32-bit BLAKE2 and to 64 bytes for 64-bit BLAKE2.

When generating hashes of unknown length (setting $\ell = 2^{32} - 1$ when using BLAKE2b, for example), depending on your use case you may need to use a salt to avoid ending up with hashes of different length such that the shorter is a prefix of the longer.

4 Performance

BLAKE2X adds a constant overhead of $\lceil \ell/64 \rceil$ (resp. $\lceil \ell/32 \rceil$) compression function calls compared to the underlying 64-bit (resp. 32-bit) BLAKE2 hash. For example, to compute a 1056-bit (132-byte) hash as required in Ed521 signatures, BLAKE2X adds $\lceil 132/3 \rceil = 3$ extra compression function calls compared to BLAKE2b. Note that $\mathsf{B2}(\mathrm{i}, \mathrm{j}, \mathrm{H}_0)$ calls can be computed in arbitrary order, and in parallel.

References

[1] Kevin Atighehchi and Alexis Bonnecaze. Asymptotic analysis of plausible tree hash modes for SHA-3. Cryptology ePrint Archive, Report 2016/658, 2016. URL: http://eprint.iacr.org/2016/658.

[2] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures - how to sign with RSA and Rabin. In *EUROCRYPT*, volume 1070 of *LNCS*, pages 399–416. Springer, 1996.

[3] Daniel J. Bernstein, Simon Josefsson, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. EdDSA for more curves. Cryptology ePrint Archive, Report 2015/677, 2015. URL: http://eprint.iacr.org/2015/677.

[4] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård revisited: How to construct a hash function. In *CRYPTO*, volume 3621 of *LNCS*, pages 430–448. Springer, 2005. Full version. URL: http://www.cs.nyu.edu/~dodis/ps/merkle.pdf.

[5] H. Krawczyk and P. Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869 (Informational), May 2010. URL: http://www.ietf.org/rfc/rfc5869.txt.

[6] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In *CRYPTO*, volume 6223 of *LNCS*, pages 631–648. Springer, 2010.

[7] Atul Luykx, Bart Mennink, and Samuel Neves. Security analysis of BLAKE2's modes of operation. Cryptology ePrint Archive, Report 2016/827, 2016. URL: http://eprint.iacr.org/2016/827.

[8] NIST. SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash. Draft Special Publication 800-185, August 2015. URL: http://csrc.nist.gov/publications/PubsDrafts.html#SP-800-185.