Lighter, Faster, and Constant-Time: WhirlBob, the Whirlpool variant of StriBob

Markku-Juhani O. Saarinen
Norwegian University of Science and Technology
mjos@item.ntnu.no

Abstract. WhirlBob is a new Authenticated Encryption with Associated Data (AEAD) algorithm derived from the first round CAESAR candidate StriBob and the Whirlpool hash algorithm. The main advantage of WhirlBob over StriBob is its greatly reduced implementation footprint on resource-constrained platforms. Remarkably, the entire C reference implementation of WhirlBob fits onto a single page of the Appendix. On most low-end microcontrollers the total software footprint of $\pi$+BLNK = WhirlBob AEAD is less than half a kilobyte. The greatly reduced hardware gate count is also reflected as efficient bitsliced straight-line implementations, especially on 64-bit platforms. Bitslicing works as an efficient countermeasure against AES-style cache timing side-channel attacks. The new design utilizes only the LPS or $\rho$ keying line of Whirlpool in a flexible domain-separated Sponge mode BLNK and adds the number of rounds in $\pi$ permutation from 10 to 12 as a countermeasure against Rebound Distinguishing attacks of ASIACRYPT '09.

As with StriBob, the reduced-size Sponge design has a strong provable security link with the original hash algorithm. We finally present some discussion and analysis on differences between Whirlpool, the Russian GOST “Streebog” hash, and the recently proposed draft Russian Encryption Standard “Kuznyechik” block cipher.

Keywords: Authenticated Encryption, Sponge designs, Whirlpool, Streebog, GOST R 34.11-2012, StriBob, CAESAR.

1 Introduction

WhirlBob is an Authenticated Encryption with Associated Data (AEAD) algorithm based on the CAESAR candidate StriBob [32, 33] and NESSIE Final Portfolio [23] hash function Whirlpool 3.0 [2]. AEAD algorithms and modes such as GCM [24] provide both confidentiality and integrity protection in a single pass, thus eliminating the requirement for an MAC algorithm such as HMAC [25]. This has clear advantages for performance and implementation footprint. Extensible-Output Functions (XOFs) such as SHA3 SHAKE [26] offer similar features to AEADs in some Sponge modes [9].

Draft document version 20140626211300.
Fig. 1. A simplified view of a Sponge-based AEAD. First the padded Secret Key, Nonce, and Associated Authenticated Data - all represented by \( d_n \) words - are “absorbed” or mixed into the Sponge state. The \( \pi \) permutation is then used to also encrypt data \( p_i \) into ciphertext \( c_i \) (or vice versa) and finally to “squeeze” out a Message Authentication Code \( h_i \).

2 Motivation: Security Goals and Parameters

WhirlBob uses StriBob’s versatile BLNK Sponge AEAD mode without modification. Outside the CAESAR context, BLNK can be also used in a wider set of applications, even to build entire secure lightweight protocol suites [31].

A sponge mode requires only a single cryptographic component; an unkeyed cryptographic permutation \( \pi \) (See Figure 1). As with other provable Sponge modes, we assume that \( \pi \) is indistinguishable from a random permutation. This work focuses on \( \pi \) permutation design – for BLNK padding details and analysis we refer to [16, 31, 33].

As it is clear that the Russian GOST hash standard Streebog [13] was closely modeled after Whirlpool [2], the only difference between StriBob and WhirlBob is in the particular numerical selections for the round constants \( C_i \), the 8-bit S-Box \( S \), byte permutation \( P \), and the \( 8 \times 8 \) MDS matrix \( L \), which is defined over a finite field \( \text{GF}(2^8) \). These components, \( L \circ P \circ S \) or the “LPS permutation” is derived almost unmodified from that of Whirlpool in present work. Both StriBob and WhirlBob have 12 rounds and the same state size.

The aim is to allow the same secure LPS implementation core (such as a special instruction of a SoC CPU in a mobile or IoT device) to be used for unkeyed hashing according to the Whirlpool standard. This is useful in applications such as certificate processing. The corresponding standardized, Miyagushi-Preneel hash functions require two (or more) times as much as state and processes data in bigger chunks. Our BLNK Sponge mode naturally also supports hashing and MACing without encryption. The Sponge variants are slightly faster.

All of the security parameters remain unmodified. As with StriBob, we have an \( b = 512 \) bit state, which is split to \( r = 256 \) -bit “block size” and \( c \approx 254 \) -bit capacity, which is the secret state. According to Theorems such as those given in [16, 33] this is sufficient for \( k = 192 \) -bit secret key security level when less than \( 2^{64} \) bits are processed under same key and nonce pair.Nonce size is largely arbitrary, but in the standard variant we adopt \( n = 128 \) bits. See Section 4 for further security analysis.
3 WhirlBob

Despite having almost equivalent speed and size on typical 64-bit platforms, the size and performance characteristics of StriBob and WhirlBob differ significantly in hardware, low-end microcontrollers, and in bitslicing implementations. We therefore suggest using WhirlBob especially in those cases.

We only give an abbreviated description of WhirlBob’s $512 \times 512$ - bit keyless $\pi$ permutation as the computation follows exactly the operation of the internal key schedule of Whirlpool 3.0 [2]. The only modification is that the number of rounds is increased from $R = 10$ to $R = 12$. The key schedule operation is also effectively equivalent to the “internal block cipher” $W$. Eight bytes from the S-Box are used as partial round keys $C_i$.

WhirlBob’s permutation $\pi$ is indeed highly similar to AES. In case of StriBob, the “Russian 512-bit block AES” permutation had to be uncovered from the structure (See Section 4.3), but the particularities and history of Whirlpool make it immediately obvious.

The 512-bit state is typically seen as an matrix of $8 \times 8$ bytes. To compute $\pi(x_0) = x_{12}$ we iterate

$$x_{i+1} = L(P(S(x_i))) \oplus C_i$$

where, if we use AES-style notation, $S$ is equivalent to SubBytes, $P$ corresponds to ShiftColumns, $L$ to MixRows, followed by AddRoundKey.

3.1 Lightweight Reference Implementation

The entire byte-oriented implementation of $\pi$ fits onto a single page; See Appendix A. Remarkably, in addition to $\pi$, only the S-Box $\text{whob}_\text{sbox}[256]$ (See Section 3.2) together with minimal BLNK logic are required for full AEAD implementation. On most microcontrollers WhirlBob’s entire software footprint is less than 500 bytes. Only slightly more is required for a shared secret handshake protocol and two-way secure BLINKER protocol [31].

This is a significant improvement over StriBob, which typically needs almost 2kB. StriBob is also much slower and larger due to the “heavy” MDS matrix. The reference implementation is not optimal when it comes to speed or size; we refer to section 7.3 of [2] for a techniques that greatly reduce the number of XORs required.

Whirlpool ISO Standard trace test vectors have been used to verify the correctness of this $\pi$ implementation, up to $R=10$. One simply observes the keying “line” of these traces and ignores the encryption “line”. We offer the listing of Appendix A as WhirlBob $\pi$ Reference implementation.

3.2 S-Box Structure and Bitsliced Implementation

Whirlpool’s S-box design utilizes three $4 \times 4$ - bit “miniboxes” given in Table 1: $E$, $E^{-1}$, and $R$. Figure 2 shows how these are used to construct the $8 \times 8$ - bit S-Box. This computation can even be performed on the fly on 4-bit microcontrollers.
Table 1. Three $4 \times 4$ miniboxes that are used to build the $8 \times 8$ S-Box.

<table>
<thead>
<tr>
<th>$x$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E(x)$</td>
<td>1</td>
<td>B</td>
<td>9</td>
<td>C</td>
<td>D</td>
<td>6</td>
<td>F</td>
<td>3</td>
<td>E</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>A</td>
<td>2</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>$E^{-1}(x)$</td>
<td>F</td>
<td>0</td>
<td>D</td>
<td>7</td>
<td>B</td>
<td>E</td>
<td>5</td>
<td>A</td>
<td>9</td>
<td>2</td>
<td>C</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>$R(x)$</td>
<td>7</td>
<td>C</td>
<td>B</td>
<td>D</td>
<td>E</td>
<td>4</td>
<td>9</td>
<td>F</td>
<td>6</td>
<td>3</td>
<td>8</td>
<td>A</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

FPGA implementations may also save a significant number of LUTs by explicitly utilizing the 4-bit structure rather than implementing a general $8 \times 8$ lookup table.

The byte-oriented $8 \times 64 = 512$ - bit state can be rapidly split into eight 64-bit registers (the first register representing the bit 0 of each one of the 64 bytes etc). This bitsliced mode can be maintained for the entire 12-round computation. Bitsliced versions of the round constants are naturally used.

The parallelism evident in Figure 2 helps to speed up bitsliced implementation. We see that for 2/3 of the time, the S-Box has effectively two independent 4-bit execution paths. Interleaving these may greatly reduce wait states due to the superscalar architecture employed by most modern CPUs. Execution time of a bitsliced software implementation not linearly dependent on the number of instructions in straight-line program.

Appendix B of current 2003 Whirlpool specification [2] gives listings with 16 ($E$ and $E^{-1}$) and 14 ($R$) instructions if the Intel standard P AND ($x \leftarrow \neg a \land b$) instruction is utilized. Better expressions are likely to exist.

Fig. 2. The $8 \times 8$ - bit S-box is constructed from $4 \times 4$ - bit “MiniBoxes”.
4 Security Analysis

We note that the security arguments and proofs offered for StriBob in [33] also apply unmodified to the new proposal, as those proofs are based on an indistinguishably arguments of the $\pi$ permutation and a simple theorem (Thm. 1, Sec. 3.3. in [33]) that loosely ties the Miyagushi-Preneel mode [22, 29] with the indistinguishably of $\pi$. A random-indistinguishable $\pi$ and appropriate padding rules are sufficient to construct Sponge-based hashes [4], Tree Hashes [8], MACs [7], Authenticated Encryption (AE) algorithms [6, 9], and pseudorandom extractors (SHAKEs, PRFs, and PRNGs) [5, 26].

4.1 Side-Channel and Implementation Attacks

Whirlpool is better suited for bitsliced implementation due to its particular S-Box and MDS design (as noted in Section 3.2). As an unconditional straight-line code without data-dependent table lookups, a bitsliced implementation is an effective countermeasure against cache timing attacks, which have been found to be effective against cryptographic primitives with large tables such as AES [1, 3, 28, 36].

A non-bitsliced implementation of the S-Box on Whirlpool, Streebog, or StriBob on 64-bit platforms typically requires lookup tables of up to $8 \times 256 \times 8 = 16384B$. Even though this size easily fits into the Level 2 cache of any 64-bit system, one may see that attacks are possible as L2 caches are not always shared even between different execution cores within a single CPU unit. This is due to the process switching operation of most 64-bit operating systems.

4.2 Historical Modifications to Whirlpool

Whirlpool has received a significant amount of analysis in the almost 15 years since its original publication. Whirlpool was the only hash function in the final NESSIE portfolio in addition to SHA-2 hashes [23]. Whirlpool has also been standardized by ISO as part of ISO/IEC 10118-3:2004 [14].

Our design is based on Whirlpool 3.0. The amended MDS matrix used by current (‘03) Whirlpool is also used by WhirlBob as a countermeasure to the structural observations given in [34].

Whirlpool was found to be vulnerable to a Rebound Distinguisher [18, 21]. That $2^{188}$ attack applies to the 10-round variant; our 12-round version should offer a comfortable security margin, especially as our security target is $2^{192}$. The way the round constants are derived from the S-Box allows this change to be made in a straightforward manner.

4.3 Notes on the origins of Streebog, Kuznyechik, and StriBob

The GOST R 34.11-2012 “Streebog” standard text [13] does not describe the linear step as a $8 \times 8$ matrix-vector multiplication with $\text{GF}(2^8)$ elements like...
the StriBob spec [33], but as a $64 \times 64$ binary matrix multiplication. One can see that $8 \times 8 \times 8 = 512$ bits are required to describe the former, but $64 \times 64 = 4096$ bits are required for the latter. The more effective description was discovered by Kazymorov and Kazymorova in [17] by exhaustively testing all 30 irreducible polynomial basis, revealing an AES-like MDS structure. The origin of the particular numerical values of that MDS matrix and round constants is still a mystery. They do not appear to offer venues for optimization like Whirlpool 3.0 and WhirlBob.

The 8-bit S-Box used by StriBob was directly lifted from Streebog so that hardware and software components developed for Streebog could be shared or recycled when implementing StriBob. The same S-Box is also used by the very recently proposed Russian Encryption Standard “Kuznyechik” block cipher [35].

Not much about the particular design criteria of the Streebog S-Box has been published. That S-box was apparently selected at least 5 years ago as Streebog already appeared in RusCrypto ’10 proceedings [20].

We can easily observe that it offers reasonable resistance against classical methods of cryptanalysis. Its differential bound [10] is $P = \frac{8}{256}$ and best linear approximation [19] holds with $P = \frac{28}{128}$. There doesn’t seem to be any exploitable algebraic weaknesses. These are the exactly same bounds as can be found for Whirlpool S-Box, but fall clearly short from the bounds of the AES S-Box.

The Rijndael AES S-box is constructed from finite field inversion $x^{-1}$ operation in $\text{GF}(2^8)$ (inspired by the Nyberg construction [27]) and an affine bit transform that serves as a countermeasure against, among other things, Interpolation Attacks [15] on AES’ predecessor SHARK [30]. We refer to [12] for more information about the AES design process.

The author had brief informal discussions with some members of the Streebog and Kuznyechik design team at the CTCrypt ’14 workshop (05-06 June 2014, Moscow RU). Their recollection was that the aim was to choose a “randomized” S-Box that meets the basic differential, linear, and algebraic requirements. Little algorithmic optimization in the S-Box generation process was used; randomization was simply iterated until a “good enough” permutation was found. This was seen as an effective countermeasure against yet-unknown attacks. At the time of Streebog S-Box selection (before 2010’s) the emergence of allegedly effective AES Algebraic Attacks such as [11] was a major concern for much of the symmetric cryptographic community. Hence it was felt appropriate to avoid too much algebraic structure in either the S-Box or MDS matrix while also ensuring necessary resistance against known attacks such as DC and LC. Algebraic attack attempts of this type against AES have since largely fizzled out, so we feel confident that the Whirlpool S-Box should be sufficient for our claimed security level, especially as it offers significantly better speeds in bitsliced implementations.

One is left with the impression that Streebog is a “whitened” or randomized copy of the original Whirlpool design. Despite its partially unknown origins and relative shortcomings on some implementation targets, we consider StriBob to be a more secure algorithm than WhirlBob if appropriately implemented. Indeed
some of the more successful attacks on AES and Whirlpool have been based on their very consistent structure and key schedule.

5 Conclusions

We have introduced the WhirlBob authenticated encryption algorithm, a variant of the StriBob first round CAESAR candidate. The new proposal loans its key components from the Whirlpool 3.0 hash function, modifying it into a Sponge AEAD. WhirlBob has extremely small implementation footprint on resource-limited software and hardware platforms—typically under half a kilobyte. The reference implementation fits onto a single page of Appendix A.

The hardware-optimized design of Whirlpool components also gives WhirlBob efficient bitsliced implementation. A bitsliced implementation is an effective countermeasure against cache timing attacks, which have been a concern against AES. The $b = 8 \times 64$-bit state size is particularly suitable for bitslicing of an byte-oriented algorithm on 64-bit platforms.

We also discussed the design choices for the S-Box and other components used in the Streebog hash and Kuznyechik cipher, which are standards or becoming standards for the Russian security market. StriBob uses this S-Box as well, and we feel that it offers better long-term security than WhirlPool, if appropriately implemented.

However WhirlBob has superb implementation characteristics on lightweight platforms and offers provable security assurance through its security reduction to the well-analyzed Whirlpool hash. Furthermore, the RAM requirement of WhirlBob AEAD is only half of that required by Whirlpool.

Acknowledgements

The author wishes to thank especially Oleksandr Kazymyrov, Vasily Shishkin, Bart Preneel, and Paulo Barreto for their helpful comments. This work was carried out during the tenure of an ERCIM “Alain Bensoussan” Fellowship Programme.

References


A WhirlBob π Byte-Oriented Reference Implementation

This ANSI C function implements the WhirlBob $512 \times 512$-bit permutation.

```c
void wbob_pi(uint8_t st[64]) // WhirlBob Pi
{
    int r, i, j;
    uint8_t t[64], x, *pt;

    for (r = 0; r < 12; r++) { // 12 rounds
        for (i = 0; i < 64; i++) {
            t[(i & 7) + ((i + (i << 3)) & 070)] = // P
                wbob_sbox[st[i]]; // S
        }
        // The round constants C comes from the S-box
        pt = (uint8_t *) &wbob_sbox[8 * r];
        for (i = 0; i < 8; i++)
            st[i] = pt[i]; // C in first 8
        for (i = 8; i < 64; i++)
            st[i] = 0; // zero the rest

        // Apply the circular, low weight MDS matrix
        for (i = 0; i < 64; i += 8) {
            pt = &st[i]; // start of row
            for (j = 0; j < 8; j++) {
                x = t[i + j]; // Circular MDS
                pt[(j & 7) & 7] ^= x; // 01
                pt[((j + 1) & 7) & 7] ^= x; // 01
                pt[((j + 3) & 7) & 7] ^= x; // 01
                pt[((j + 5) & 7) & 7] ^= x;
                pt[((j + 7) & 7) & 7] ^= x;
                // x <- 02
                x = (x << 1) ^ (x & 0x80 ? 0x1D : 0x00);
                pt[((j + 6) & 7) & 7] ^= x; // 02
                // x <- 04
                x = (x << 1) ^ (x & 0x80 ? 0x1D : 0x00);
                pt[((j + 2) & 7) & 7] ^= x; // 04
                pt[((j + 5) & 7) & 7] ^= x; // 01 + 04 = 05
                // x <- 08
                x = (x << 1) ^ (x & 0x80 ? 0x1D : 0x00);
                pt[((j + 4) & 7) & 7] ^= x; // 08
                pt[((j + 7) & 7) & 7] ^= x; // 01 + 08 = 09
            }
        }
    }
}
```