WHIRLBOB, the Whirlpool Based Variant of STRIBOB

Lighter, Faster, and Constant Time

Markku–Juhani O. Saarinen[1(✉)] and Billy Bob Brumley[2]

[1] Centre for Secure Information Technologies (CSIT) ECIT, Queen's University Belfast, Belfast, UK
m.saarinen@qub.ac.uk

[2] Tampere University of Technology, Tampere, Finland
billy.brumley@tut.fi

Abstract. WHIRLBOB, also known as STRIBOBr2, is an AEAD (Authenticated Encryption with Associated Data) algorithm derived from STRIBOBr1 and the Whirlpool hash algorithm. WHIRLBOB/STRIBOBr2 is a second round candidate in the CAESAR competition. As with STRIBOBr1, the reduced-size Sponge design has a strong provable security link with a standardized hash algorithm. The new design utilizes only the LPS or ρ component of Whirlpool in flexibly domain-separated BLNK Sponge mode. The number of rounds is increased from 10 to 12 as a countermeasure against Rebound Distinguishing attacks. The 8×8 - bit S-Box used by Whirlpool and WHIRLBOB is constructed from 4×4 - bit "MiniBoxes". We report on fast constant-time Intel SSSE3 and ARM NEON SIMD WHIRLBOB implementations that keep full miniboxes in registers and access them via SIMD shuffles. This is an efficient countermeasure against AES-style cache timing side-channel attacks. Another main advantage of WHIRLBOB over STRIBOBr1 (and most other AEADs) is its greatly reduced implementation footprint on lightweight platforms. On many lower-end microcontrollers the total software footprint of π+BLNK = WHIRLBOB AEAD is less than half a kilobyte. We also report an FPGA implementation that requires 4,946 logic units for a single round of WHIRLBOB, which compares favorably to 7,972 required for Keccak / Keyak on the same target platform. The relatively small S-Box gate count also enables efficient 64-bit bitsliced straight-line implementations. We finally present some discussion and analysis on the relationships between WHIRLBOB, Whirlpool, the Russian GOST Streebog hash, and the recent draft Russian Encryption Standard Kuznyechik.

Keywords: WHIRLBOB · STRIBOBr1 · Authenticated encryption · Sponge designs · Timing attacks · Whirlpool · Streebog · CAESAR competition

Much of this research was carried out during tenure of an ERCIM "Alain Bensoussan" Fellowship Programme at NTNU, Trondheim.

S. Buchegger and M. Dam (Eds.): NordSec 2015, LNCS 9417, pp. 106–122, 2015.
DOI: 10.1007/978-3-319-26502-5_8

1 Introduction

STRIBOBr2, or WHIRLBOB, is an Authenticated Encryption with Associated Data (AEAD) algorithm and a CAESAR ("Competition for Authenticated Encryption: Security, Applicability, and Robustness") [21] competition Second Round Candidate [58].

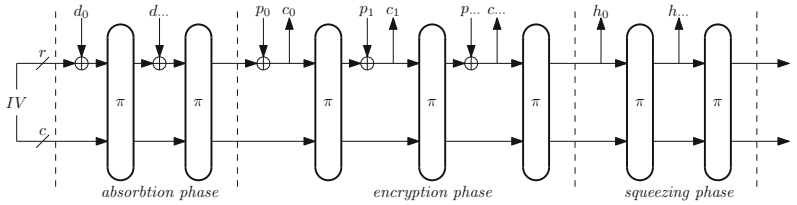

Fig. 1. A simplified view of a Sponge-based AEAD. Padded Secret Key, Nonce, and Associated Authenticated Data - all represented by d_u words - are first "absorbed" into the state. The π permutation is then also used to encrypt data p_i into ciphertext c_i (or vice versa) and finally to "squeeze" out a Message Authentication Code h_i.

AEAD algorithms and modes such as GCM [45] provide both confidentiality and integrity protection for messages in a single step, thus eliminating the need for a separate MAC algorithm such as HMAC [46]. This has clear advantages for performance and implementation footprint.

WHIRLBOB uses STRIBOBr1's BLNK Sponge AEAD mode and parameters 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 [53]. A sponge mode requires only a single cryptographic component; an unkeyed cryptographic permutation π (See Figure 1). As with other provable Sponge modes, we assume that π is indistinguishable from a random permutation. This work focuses on π permutation design and implementation – for BLNK padding details and analysis we refer to [32,54,56,57].

The STRIBOBr1 CAESAR [56] candidate was derived from the Russian GOST hash standard Streebog [26]. In close examination Streebog appears to be modeled after the Whirlpool hash [4], with substantial modifications. However, STRIBOBr1 and WHIRLBOB only differ in the particular numerical selections for tables C, S, and L. These components, $L \circ P \circ S$ or the "LPS permutation" is derived directly from that of Whirlpool for WHIRLBOB. The program code of 64-bit reference implementations is essentially equivalent for both algorithms. Both STRIBOBr1 and WHIRLBOB have 12 rounds.

We show that the particular structure of the Whirlpool components allows WHIRLBOB to have much more efficient SIMD, constant-time, lightweight, and hardware implementations. One of our aims is to allow the same implementation core (such as a special instruction or coprocessor of a SoC [56]) to be also used for unkeyed hashing according to the Whirlpool standard. This is useful in

applications that also require efficient standards-compliant certificate signature processing.

The corresponding standardized, Miyagushi-Preneel hash functions Streebog and Whirlpool require two (or more) times as much as state and processes data in bigger chunks when compared to STRIBOBr1 and WHIRLBOB. Our BLNK Sponge mode also supports randomized hashing and MACing without encryption. Our Sponge variants are slightly faster than the original hashes, yet have a provable security relation. All security parameters remain unmodified from STRIBOBr1.

2 WHIRLBOB

As with STRIBOBr1, the state consists of $b = 512$ bits, split in our BLNK mode as:

$r = 256$ - bit *rate* "block size", which directly interacts with output and input.
$c \approx 254$ - bit *capacity*, which is the secret state. Some bits are lost to padding.

These two halves are mixed together by a keyless, random-indistinguishable permutation π. According to theorems such as those given in [32,56] 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. For the CAESAR variant the nonce length is fixed at $n = 128$ bits.

Despite having almost equivalent speed and size on generic 64-bit platforms, the size and performance characteristics of STRIBOBr1 and WHIRLBOB differ significantly on various implementation platforms such as FPGA, low-end microcontrollers, SIMD systems, and in bitslicing implementations.

WHIRLBOB's permutation π is indeed highly similar to AES. In case of STRIBOBr1, the "Russian 512-bit block AES" permutation had to be somewhat laboriously uncovered from the structure (see Section 5.3), but the particularities and history of Whirlpool make the connection immediately clear.

2.1 Structure of the π Permutation

The computation of π follows almost exactly the operation of the internal key schedule of Whirlpool 3.0 [4]. The only modification is that the number of rounds is increased from $R = 10$ to $R = 12$ for extra security margin against Rebound Distinguisher attacks [35,36]. [1]

To compute $\pi(x_0) = x_{12}$ we iterate the $\mathsf{LPS} = L \circ P \circ S$ composite mixing function with round constants C_r. For rounds $0 \leq r < 12$:

$$x_{r+1} = L(P(S(x_r))) \oplus C_r. \tag{1}$$

[1] These attacks would not be directly applicable with the BLNK mode anyway. This is because the attacker can never access more than r bits of the internal state.

If we use the AES-style notation of Whirlpool, S is equivalent to `SubBytes`, P corresponds to `ShiftColumns`, L to `MixRows`, followed by `AddRoundKey`.

We write the 512-bit state as a matrix $M[\ 0 \cdots 7\ ][\ 0 \cdots 7\ ]$ of 8×8 bytes, which can be serialized to a byte vector as $\texttt{vec}[\ 8i + j\ ] = M[\ i\ ][\ j\ ]$.

S: `SubBytes`: Each one of the 64 bytes in the state is substituted using the (singular) 8×8 - bit S-Box described in Section 2.2. For $0 \leq i, j < 8$

$$M'[\ i\ ][\ j\ ] \leftarrow S(M[\ i\ ][\ j\ ]). \tag{2}$$

P: `ShiftColumns`: A byte shuffle. For $0 \leq i, j < 8$

$$M'[\ (\ i + j\ ) \bmod 8\ ][\ j\ ] \leftarrow M[\ i\ ][\ j\ ]. \tag{3}$$

L: `MixRows`: Each of the 8 row vectors

$$V_i = (\ M[\ i\ ][\ 0\ ], M[\ i\ ][\ 1\ ], \cdots M[\ i\ ][\ 7\ ]\ ) \tag{4}$$

is individually multiplied by a circulant, low-weight 8×8 MDS matrix in the finite field $\mathsf{GF}(2^8)$ characterized by primitive polynomial $p(x) = x^8 + x^4 + x^3 + x^2 + 1$.

$$V_i' = V_i \cdot \begin{pmatrix} \texttt{01 01 04 01 08 05 02 09} \\ \texttt{09 01 01 04 01 08 05 02} \\ \texttt{02 09 01 01 04 01 08 05} \\ \texttt{05 02 09 01 01 04 01 08} \\ \texttt{08 05 02 09 01 01 04 01} \\ \texttt{01 08 05 02 09 01 01 04} \\ \texttt{04 01 08 05 02 09 01 01} \\ \texttt{01 04 01 08 05 02 09 01} \end{pmatrix}. \tag{5}$$

C_r: `AddRoundKey`: The key schedule operation is effectively equivalent to the one used by Whirlpool's "internal block cipher" W. Blocks of eight bytes from the S-Box are used round keys C_r for the first row. For round $0 \leq r < 12$, $0 \leq j < 8$:

$$M'[\ 0\ ][\ j\ ] \leftarrow M[\ 0\ ][\ j\ ] \oplus S(\ 8r + j\ ) \tag{6}$$

Rest of the rows are unaffected by C_r. For $1 \leq i < 8, 0 \leq j < 8$:

$$M'[\ i\ ][\ j\ ] \leftarrow M[\ i\ ][\ j\ ]. \tag{7}$$

We offer the listing of Appendix A as WHIRLBOB v1.0 π reference implementation. Whirlpool ISO Standard trace test vectors can be used to verify the correctness of this π implementation up to $r = 10$ [30]. One simply observes the keying "line" of these traces and ignores the encryption "line".

2.2 The S-Box

The Whirlpool and WHIRLBOB 8×8-bit S-Box design utilizes three 4×4 - bit "miniboxes" given in Table 1: E, E^{-1}, and R. Figure 2 shows how these are used to construct the 8×8 - bit S-Box.

Table 1. Three 4×4 miniboxes that are used to build the 8×8 S-Box in Whirlpool 3.0 and WHIRLBOB 1.0.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$E(x)$	1	B	9	C	D	6	F	3	E	8	7	4	A	2	5	0
$E^{-1}(x)$	F	0	D	7	B	E	5	A	9	2	C	1	3	4	8	6
$R(x)$	7	C	B	D	E	4	9	F	6	3	8	A	2	5	1	0

This computation can even be performed on the fly on 4-bit microcontrollers. FPGA implementations save a significant number of LUTs by explicitly utilizing the 4-bit structure rather than implementing a general 8×8 lookup table. These small S-Boxes can often fit into SIMD registers and accessed via constant-time shifts or shuffles, thus enabling implementations resistant to timing attacks.

2.3 BLNK Mode

The padding details and operation of BLNK Sponge mode for WHIRLBOB and STRIBOBr1 are equivalent. Please see [58] for details. The mode is based derived from the Blinker light-weight protocol [53], but limited to CAESAR use case.

3 Implementation

The entire byte-oriented implementation of π fits onto a single page; See Appendix A. Remarkably, in addition to π, only the S-Box `wbob_sbox[256]` (See Section 2.2) together with minimal BLNK logic are required for full AEAD implementation. On many microcontrollers WHIRLBOB's entire software footprint is in the 500B range. Slightly more is required for a shared secret handshake protocol and two-way secure BLINKER protocol [55].

This is a significant improvement over STRIBOBr1, which typically needs almost 2kB. STRIBOBr1 is also much slower and larger on low-end microcontrollers due to the "heavy" MDS matrix. The reference implementation is written for compactness and clarity; it is not optimal when it comes to speed or size. We refer to section 7.3 of [4] for techniques that greatly reduce the number of XORs required, resulting in increased processing speed. Additional tables will be required, however, and this will increase the overall implementation size.

3.1 Constant-Time SIMD Implementation

Due largely to Whirlpool's S-Box structure and generous parallelism, it is well suited for high speed, constant-time implementation on Single Instruction Multiple Data (SIMD) architectures. Here we focus on ARM's NEON as the reference architecture since the state layout fits the registers nicely, but also consider Intel's SSSE3 as another explicit example. The goal is to improve performance, while at the same time avoiding memory-resident table lookups that cause execution

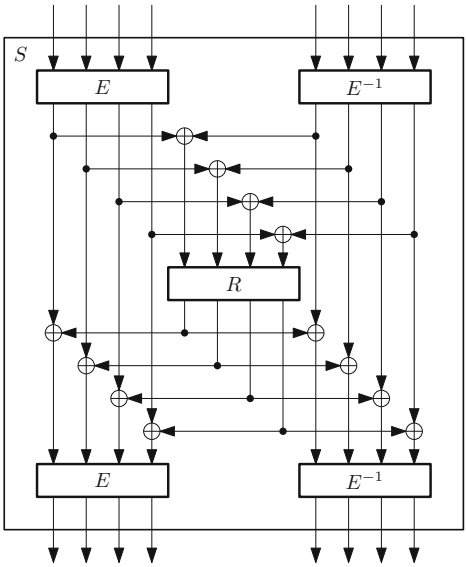

Fig. 2. The WHIRLBOB 8×8 - bit S-Box is constructed from three 4×4 - bit "miniboxes". In this diagram the most significant bits are on the left: E operates on the higher nibble.

time to depend on the data cache state and thus algorithm state (the crux of cache timing attacks).

Related work in this area includes simulated ISA extensions to a RISC architecture for parallel table lookups to speed up Whirlpool [28]. These extensions are then used to build essentially a hardware-assisted analogue of the traditional T tables software implementation – storing the state in rows and issuing a single instruction to perform 8 parallel lookups from the 8-bit S-Box input to the 64-bit linear layer output and XOR-summing the results, repeated for each row. AES [27] and Anubis [18] can also take advantage of SSSE3's variable byte shuffle instruction for fast and secure implementations.

NEON has 32×64 - bit SIMD registers and SSSE3 16×128 - bit. We store the state column-wise (one column per NEON register, two columns per SSSE3 register), i.e. byte position j of register i contains the state byte in column i and row j. The `SubBytes` step is not sensitive to this ordering, but both `ShiftColumns` and `MixRows` are. Since both of these architectures feature variable byte shuffle instructions (`vtbl.u8` for NEON and `pshufb` for SSSE3), implementing `SubBytes` is a direct translation of Figure 2 to these instructions. This amounts to 40 NEON shuffles and half as many SSSE3 shuffles. For `ShiftColumns`, NEON uses `vext` for byte-wise register rotation and SSSE3 `pshufb` with constant rotation distances since each register holds two columns. For `MixRows` we use the row formula from the Whirlpool specification [4, Sec. 7.3] where the multiplications by x are a simple left shift (native on NEON,

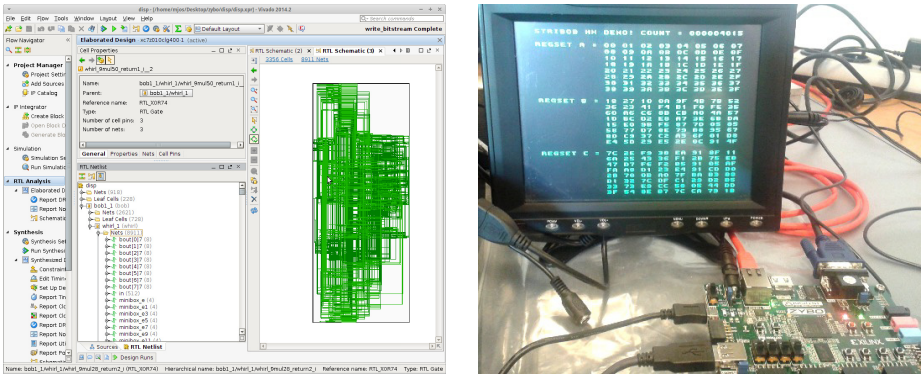

Fig. 3. WHIRLBOB was implemented on the FPGA logic fabric of Xilix Zynq 7010. The implementation integrates with the AXI bus of ARM Cortex A9 on the SoC chip.

integer addition on SSSE3) and conditional XOR (operand masked by signed right shift on NEON, comparison on SSSE3). The formula is fairly symmetric around even and odd byte positions – while NEON implements it as written with 24 multiplications, SSSE3 slightly rearranges a few registers to parallelize across the full 128-bit register width and use half as many multiplications.

3.2 Generic Constant-Time Bitsliced Implementation

The byte-oriented $8 \times 64 = 512$ - bit state can be rapidly split into eight 64-bit registers. 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.

Appendix B of the current 2003 Whirlpool specification [4] gives listings with 14-16 instructions/gates for each of the miniboxes (if ANDN instruction is allowed). Those were used in our reference bitsliced implementations.

3.3 Implementation Summary

We currently have six implementations of WHIRLBOB. They mainly differ in the implementation technique used for the π cryptographic permutation.

- **C 8-bit**: This is the minimal reference implementation which is optimized for clarity and low-resource platforms, corresponding to Appendix A.
- **C 64-bit**: Standard speed-optimized implementation for most platforms, utilizing large lookup tables. Apart from Whirlpool-derived tables, equivalent to the implementation of [56].
- **C Bitsliced**: Straight-line, fully bitsliced implementation without data-dependent branches or lookups. Resistant to timing attacks.

- **NEON Intrinsics**: Fast constant-time version that avoids table lookups by storing 4×4 - bit miniboxes in SIMD registers.
- **SSSE3 Intrinsics**: Similar but for 128-bit SIMD registers.
- **Verilog 12-cycle**: This is the hardware reference implementation. Source code is about 350 lines. Additional logic is required for AXI Bus integration.

Software Implementations. The first three implementations use only C99, and are hence easily portable. See Table 2 for implementation metrics. We also have various embedded implementations. Note that STRIBOBr2 is faster then the (out-of-box, Ubuntu 14.04 LTS) OpenSSL implementation of AES-192 on the same target.

Hardware Implementation. The hardware implementation has been proven on FPGA (Figure 3). The SÆHI proposal reports total post place-and-route utilization on Artix-7 of 4,946 logic units for a single round of WHIRLBOB, which compares favourably to 7,972 required for Keccak/Keyak [56]. Throughput is roughly 2 MB/s for each MHz.

Table 2. Comparing software implementations of WHIRLBOB π.

Target	**Speed**	**Footprint**		**Source**
	MB/sec	Code	Data	C lines
Single Core of 3.4GHz Core i7-4770				
8-bit C99 Reference	7.772	326	256	97
Bitsliced C99 Reference	49.02	4592	768	345
64-bit C99 Reference	139.2	1942	16512	128
SSSE3 (Constant-Time)	**162.3**	1290	1152	256
OpenSSL 1.0.1f AES-192 CBC	*145.6*			
BeagleBone Black 1.0GHz Cortex-A8				
8-bit C99 Reference	0.828	352	256	97
64-bit C99 Reference	3.343	6524	16512	128
Bitsliced C99 Reference	1.435	15704	768	345
NEON (Constant-Time)	9.208	1528	1072	320

4 Comparison with Other AEAD Schemes

At the time of writing the dominant AEAD scheme is the Galois / Counter Mode (GCM) for the AES block cipher [43,45], which is recommended for use with TLS, SSH and IPSec protocols by NSA as part of "Suite B" [19,29,48,59]. GCM message authentication is based on polynomial evaluation in the finite field $\mathsf{GF}(2^{128})$. The required multiplication can be exceedingly slow on lightweight platforms. An LFSR-style implementation of a 128×128 - bit multiplication will require thousands of cycles on 8-bit targets.

It is often more efficient to use the CCM [44,63] double-mode of operation on lightweight platforms, since implementing a full extra AES operation can be

Table 3. Relative performance of some CAESAR candidates on a AMD64 reference system in SUPERCOP testing (smaller number indicates faster speed).

MORUS 1280 - 128 [64]	0.09
NORX 64-4-1 [3]	0.19
ASCON-128n [24]	0.89
WHIRLBOB Intel SSSE3 Constant-Time	**1.00**
WHIRLBOB and STRIBOBr1 64-bit Reference	1.26
Lake Keyak [12]	2.23
Ketje Sr. [6]	4.25
PRIMATES (HANUMAN, GIBBON, APE) [2]	50+

bench.cr.yp.to/web-impl/amd64-titan0-crypto_aead.html

faster than the finite field multiplication operation. CCM and GCM are currently the only two FIPS - standardized authenticated modes. The performance characteristics of AES-CCM AEAD can be expected to be very similar to WHIRLBOB due to their structural similarities and relative data bandwidth:

- WHIRLBOB: 12 rounds with 64 S-Boxes for 256 bits of data.
- AES-192-CCM: 2×12 rounds with 16 S-Boxes for 128 bits of data.

There are additional (patented) AES modes which will be faster on 8-bit platforms – such as AES-OTR [40] and AES-OCB [34], and dozens of others. Virtually all AEAD block cipher modes offer lower levels of integrity protection (2^{64} level even for 128-bit tags) and are not directly usable in wider Sponge applications such as non-randomized hashing.

Currently only unoptimized reference implementations are available for most CAESAR candidates [20], making fair performance comparisons difficult. Furthermore, no other CAESAR candidate is targeted at 192-bit security level (apart from AES modes) and little attention has been paid to 8-bit or hardware implementations.

We note that leading full-featured Sponge candidates, directly SHA3 / Keccak - based Ketje Keyak [12] and [6] have significantly slower reference implementation than STRIBOBr1 and WHIRLBOB (Table 3). WHIRLBOB falls very significantly from candidates such as NORX [3] and MORUS [64], which have been designed specifically with 64-bit targets in mind. Our proposal can claim a more conservative security margin when compared to these candidates, however.

5 Security Analysis and Design Notes

For analysis of the round function against classical Differential and Linear cryptanalysis we refer to Whirlpool literature [4]. Two additional rounds increase WHIRLBOB's resistance against best known attacks [35,36].

Most of the security arguments and proofs offered for STRIBOBr1 and BLNK in [56] also apply to the new proposal. These are based on indistinguishably arguments for the π permutation and a simple theorem (Thm. 1, Sec. 3.3. in [56]) that loosely ties the compression function in Miyagushi-Preneel mode [41,51] with the indistinguishably of π. A random-indistinguishable π and appropriate padding rules are sufficient to construct Sponge-based hashes [7], Tree Hashes [11], MACs [10], Authenticated Encryption (AE) algorithms [9], and pseudorandom extractors (SHAKEs, PRFs, and PRNGs) [8,47].

5.1 Side-Channel and Implementation Attacks

Due to the minibox structure, we may load the 4×4 - bit tables in registers and access them via constant-time shuffles on Intel SSSE3 and ARM NEON SIMD targets as noted in Section 3.1. WHIRLBOB is also relatively well suited for bitsliced implementation due to its particular S-Box and MDS design as noted in Section 2.2.

Being unconditional straight-line code without data-dependent table lookups, bitsliced and byte shuffling implementations are effective countermeasures against cache timing attacks, which can be mounted against cryptographic primitives with large tables such as AES [1,5,50,62].

A non-constant-time implementation of the S-Box on Whirlpool, Streebog, or STRIBOBr1 on 64-bit platforms typically requires lookup tables of up to $8 \times 256 \times 8 = 16384$B. Even though this size easily fits into the Level 2 cache of any 64-bit system, one may see that timing 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.

5.2 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 [42]. Whirlpool has also been standardized by ISO as part of ISO/IEC 10118-3:2004 [30].

The amended MDS matrix used by current ('03) Whirlpool is also used by WHIRLBOB as a countermeasure to the structural observations given in [60]. Our design is based on Whirlpool 3.0.

Whirlpool was found to be vulnerable to a Rebound Distinguisher [35,36,39]. 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.

5.3 Notes on the Origins of Streebog, Kuznyechik, and STRIBOBr1

The 8-bit S-Box used by STRIBOBr1 was directly lifted from Streebog so that hardware and software components developed for Streebog could be shared or

recycled when implementing STRIBOBr1. The same S-Box is also used by the recently proposed Russian Encryption Standard "Kuznyechik" [25,61].

The GOST R 34.11-2012 "Streebog" standard text [26] does not describe the linear step as a 8×8 matrix-vector multiplication with $\mathsf{GF}(2^8)$ elements like the STRIBOBr1 spec [56], but as a 64×64 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 [33] by exhaustively testing all 30 irreducible polynomial bases, revealing an AES-like MDS structure. The origin of the particular numerical values of that MDS matrix is still a mystery. They do not appear to offer similar avenues for size or performance optimization like those in Whirlpool 3.0 and STRIBOBr2 do.

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 [38]. Very recent ongoing work has revealed it to also have an optimized representation [16], after all.

We can easily observe that the S-Box offers reasonable resistance against basic methods of cryptanalysis. Its differential bound [13] is $P = \frac{8}{256}$ and best linear approximation [37] holds with $P = \frac{28}{128}$. There does not seem to be any exploitable algebraic weaknesses. These are the exact same bounds as can be found for the Whirlpool and STRIBOBr2 S-Box, but fall short from the bounds of the AES S-Box.

By comparison, the Rijndael AES S-Box is constructed from finite field inversion x^{-1} operation in $\mathsf{GF}(2^8)$ (inspired by the Nyberg construction [49]) and an affine bit transform that serves as a countermeasure against, among other things, Interpolation Attacks [31] on the AES predecessor SHARK [52]. We refer to [23] for more information about the AES design process.

We 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. Randomization using various building blocks 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 [22] 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. We feel confident that the Whirlpool S-Box should be sufficient for our claimed security level, especially as it offers significantly better speeds in constant-time implementations when compared to an AES-Style S-Box.

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 STRIBOBr1 to be at least as secure as STRIBOBr2 if appropriately implemented. Indeed some of the more successful attacks on AES and Whirlpool have been based on their deep structural self-similarities and simplistic key schedules [14,15,17], so STRIBOBr1 may have some security advantages against "unknown" attacks.

6 Conclusions

We have introduced WHIRLBOB, an algorithm for Authenticated encryption with Associated Data. WHIRLBOB is a variant of the STRIBOBr1 first round CAESAR candidate but borrows its main components from the Whirlpool 3.0 hash. WHIRLBOB, also known as STRIBOBr2, is a CAESAR [21] second round candidate [58].

WHIRLBOB has extremely small implementation footprint on resource-limited software platforms – typically under half a kilobyte. Its particular S-Box and MDS design allows WHIRLBOB to have efficient constant-time bitsliced and SIMD byte shuffling implementations. This is an effective countermeasure against cache timing attacks, which are a concern against AES. The $b = 8 \times 64$ - bit state size is particularly suitable for bitslicing of a byte-oriented algorithm on 64-bit platforms and byte slicing for SIMD platforms.

WHIRLBOB has superb implementation characteristics on FPGA (ASIC), SIMD and lightweight embedded platforms. We recommend WHIRLBOB especially for those platforms. Furthermore WHIRLBOB offers provable security assurance through its security relationship with the well-analyzed Whirlpool hash.

We have also discussed the design choices for the STRIBOBr1 S-Box and other components used in the Streebog hash and Kuznyechik cipher, which are becoming standards for the Russian security market.

Acknowledgments. We thank Oleksandr Kazymyrov, Vasily Shishkin, Bart Preneel, and Paulo Barreto for their helpful comments.

Supported in part by TEKES grant 4681/31/2014 INKA EAKR Hardware Rooted Security. This article is based upon work from COST Action CRYPTACUS, supported by COST (European Cooperation in Science and Technology).

References

1. Acıiçmez, O., Schindler, W., Koç, Ç.K.: Cache based remote timing attack on the aes. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 271–286. Springer, Heidelberg (2006)
2. Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: PRIMATEs v1 - Submission to the CAESAR Competition. CAESAR First Round Submission, March 2014. http://competitions.cr.yp.to/round1/primatesv1.pdf

3. Aumasson, J.-P., Jovanovic, P., Neves, S.: CAESAR submission: NORX v1. CAESAR First Round Submission, March 2014. http://competitions.cr.yp.to/round1/norxv1.pdf
4. Barreto, P.S.L.M., Rijmen, V.: The Whirlpool hashing function. NESSIE Algorithm Specification, 2000, revised May 2003. http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html
5. Bernstein, D.J.: Cache-timing attacks on AES. Technical report, University of Chigaco, 2005. http://cr.yp.to/antiforgery/cachetiming-20050414.pdf
6. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: CAESAR submission: Ketje v1. CAESAR First Round Submission, March 2014. http://competitions.cr.yp.to/round1/ketjev1.pdf
7. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge functions. In: Ecrypt Hash Workshop 2007, May 2007. http://events.iaik.tugraz.at/HashWorkshop07/program.html
8. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge-based pseudo-random number generators. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 33–47. Springer, Heidelberg (2010)
9. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Duplexing the sponge: single-pass authenticated encryption and other applications. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 320–337. Springer, Heidelberg (2012)
10. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The Keccak reference, version 3.0. NIST SHA3 Submission Document, January 2011. http://keccak.noekeon.org/Keccak-reference-3.0.pdf
11. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sakura: a flexible coding for tree hashing. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS 2014. LNCS, vol. 8479, pp. 217–234. Springer, Heidelberg (2014)
12. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: CAESAR submission: Keyak v1. CAESAR First Round Submission, March 2014. http://competitions.cr.yp.to/round1/keyakv1.pdf
13. Biham, E., Shamir, A.: Differential Cryptanalysis of the Data Encryption Standard. Springer (1993)
14. Biryukov, A., Khovratovich, D.: Related-key cryptanalysis of the full AES-192 and AES-256. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 1–18. Springer, Heidelberg (2009)
15. Biryukov, A., Khovratovich, D., Nikolić, I.: Distinguisher and related-key attack on the full AES-256. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 231–249. Springer, Heidelberg (2009)
16. Biryukov, A., Perrin, L., Udovenko, A.: The secret structure of the S-Box of Streebog, Kuznechik and StriBob. IACR ePrint 2015/812, August 2015. https://eprint.iacr.org/2015/812
17. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique cryptanalysis of the full AES. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 344–371. Springer, Heidelberg (2011)
18. Brumley, B.B.: Secure and fast implementations of two involution ciphers. In: Aura, T., Järvinen, K., Nyberg, K. (eds.) NordSec 2010. LNCS, vol. 7127, pp. 269–282. Springer, Heidelberg (2012)
19. Burgin, K., Peck, M.: Suite B Profile for Internet Protocol Security (IPsec). IETF RFC **6380**, October 2011
20. CAESAR. CAESAR: Competition for authenticated encryption: Security, applicability, and robustness, January 2014. http://competitions.cr.yp.to/caesar.html

21. CAESAR. CAESAR first and second round submissions, July 2015. http://competitions.cr.yp.to/caesar-submissions.html
22. Courtois, N.: How fast can be algebraic attacks on block ciphers? IACR ePrint 2006/168, May 2006. https://eprint.iacr.org/2006/168
23. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - the Advanced Encryption Standard. Springer (2002)
24. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1 - Submission to the CAESAR Competition. CAESAR First Round Submission, March 2014. http://competitions.cr.yp.to/round1/asconv1.pdf
25. Dygin, D.M., Lavrikov, I.V., Marshalko, G.B., Rudskoy, V.I., Trifonov, D.I., Shishkin, V.A.: On a new Russian Encryption Standard. Mathematical Aspects of Cryptography **6**(2), 29–34 (2015). http://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=mvk&paperid=142&option_lang=eng (Abstract In Russian)
26. GOST. Information technology. cryptographic protection of information, hash function. GOST R 34.11-2012 (2012). http://protect.gost.ru/v.aspx?control=7&id=180209 (In Russian)
27. Hamburg, M.: Accelerating AES with vector permute instructions. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 18–32. Springer, Heidelberg (2009)
28. Hilewitz, Y., Yin, Y.L., Lee, R.B.: Accelerating the whirlpool hash function using parallel table lookup and fast cyclical permutation. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 173–188. Springer, Heidelberg (2008)
29. Igoe, K.: Suite B Cryptographic Suites for Secure Shell (SSH). IETF RFC 6239, May 2011. https://tools.ietf.org/html/rfc6239
30. ISO/IEC. Information technology - security techniques - hash-functions - part 3: Dedicated hash-functions. ISO/IEC 10118–3:2004 (2004). https://www.iso.org/obp/ui/#iso:std:iso-iec:10118:-3:ed-3:v1:en
31. Jakobsen, T., Knudsen, L.R.: The interpolation attack on block ciphers. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 99–112. Springer, Heidelberg (1997)
32. Jovanovic, P., Luykx, A., Mennink, B.: Beyond $2^{c/2}$ security in sponge-based authenticated encryption modes. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 85–104. Springer, Heidelberg (2014)
33. Kazymyrov, O., Kazymyrova, V.: Algebraic aspects of the Russian hash standard GOST R 34.11-2012. In: CTCrypt 2013, June 23–24, 2013, Ekaterinburg, Russia, 2013. IACR ePrint 2013/556. https://eprint.iacr.org/2013/556
34. Krovetz, T., Rogaway, P.: OCB (v1). CAESAR First Round Submission, March 2014. http://competitions.cr.yp.to/round1/ocbv1.pdf
35. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound distinguishers: results on the full whirlpool compression function. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 126–143. Springer, Heidelberg (2009)
36. Lamberger, M., Mendel, F., Schläffer, M., Rechberger, C., Rijmen, V.: The rebound attack and subspace distinguishers: Application to Whirlpool. J. Cryptology **28**, 257–296 (2015)
37. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
38. Matyuhin, D.V., Rudskoy, V.I., Shishkin, V.A.: Promising hashing algorithm. RusCrypto 2010. Workshop **02**, 2010 (2010). (In Russian)
39. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The rebound attack: cryptanalysis of reduced Whirlpool and `Grøstl`. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 260–276. Springer, Heidelberg (2009)
40. Minematsu, K.: AES-OTR v1. CAESAR First Round Submission, March 2014. http://competitions.cr.yp.to/round1/aesotrv1.pdf

41. Miyaguchi, S., Ohta, K., Iwata, M.: 128-bit hash function (n-hash). NTT Review **2**, 128–132 (1990)
42. NESSIE. Final report of European project number IST-1999-12324, named New European Schemes for Signatures, Integrity, and Encryption. NESSIE, April 2004. https://www.cosic.esat.kuleuven.be/nessie/Bookv015.pdf
43. NIST. Advanced Encryption Standard (AES). Federal Information Processing Standards Publication FIPS 197, November 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
44. NIST. Counter with Cipher Block Chaining - Message Authentication Code (CCM). NIST Special Publication 800–38C, May 2004
45. NIST. Recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC. NIST Special Publication 800–38D (2007). http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf
46. NIST. The Keyed-Hash Message Authentication Code (HMAC). Federal Information Processing Standards Publication FIPS 198–1, July 2008
47. NIST VCAT. NIST Cryptographic Standards and Guidelines Development Process: Report and Recommendations of the Visiting Committee on Advanced Technology of the National Institute of Standards and Technology, July 2014
48. NSA. Suite B Cryptography (2005). http://www.nsa.gov/ia/programs/suiteb_cryptography
49. Nyberg, K.: Differentially uniform mappings for cryptography. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 55–64. Springer, Heidelberg (1994)
50. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: the case of AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006)
51. Preneel, B.: Analysis and Design of Cryptographic Hash Functions. PhD thesis, K. U. Leuven (Belgium) (1993). http://homes.esat.kuleuven.be/~preneel/phd_preneel_feb1993.pdf
52. Rijmen, V., Daemen, J., Preneel, B., Bosselaers, A., De Win, E.: The cipher SHARK. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 99–111. Springer, Heidelberg (1996)
53. Saarinen, M.-J.O.: Beyond modes: building a secure record protocol from a cryptographic sponge permutation. In: Benaloh, J. (ed.) CT-RSA 2014. LNCS, vol. 8366, pp. 270–285. Springer, Heidelberg (2014)
54. Saarinen, M.-J.O.: Simple AEAD hardware interface (SÆHI) in a SoC: implementing an on-chip Keyak/WhirlBob coprocessor. In: TrustED 2014 Proceedings of the 4th International Workshop on Trustworthy Embedded Device, pp. 51–56. ACM (2014)
55. Saarinen, M.-J.O.: StriBob: Authenticated encryption from GOST R 34.11-2012 LPS permutation. In: Preproceedings of the CTCrypt 2014, 05–06 June 2014, Moscow, Russia, pp. 170–182, June 2014. https://eprint.iacr.org/2014/271
56. Saarinen, M.-J.O.: The STRIBOBr 1 authenticated encryption algorithm. CAESAR, 1st Round Candidate, March 2014. http://www.stribob.com
57. Saarinen, M.-J.O.: StriBob: authenticated encryption from GOST R 34.11-2012 LPS permutation. Mathematical Aspects of Cryptography **6**(2), 67–78 (2015). http://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=mvk&paperid=146&option_lang=eng (Abstract In Russian)
58. Saarinen, M.-J.O., Brumley, B.B.: STRIBOBr 2: "WHIRLBOB", second round caesar algorithm tweak specification. CAESAR 2nd Round Candidate, August 2015. http://www.stribob.com

59. Salter, M., Housley, R.: Suite B Profile for Transport Layer Security (TLS). IETF RFC 6460, January 2012. https://tools.ietf.org/html/rfc6460
60. Shirai, T., Shibutani, K.: On the diffusion matrix employed in the Whirlpool hashing function. NESSIE Public Report (2003). http://www.cosic.esat.kuleuven.be/nessie/reports/phase2/whirlpool-20030311.pdf
61. Shishkin, V., Dygin, D., Lavrikov, I., Marshalko, G., Rudskoy, V., Trifonov, D.: Low-weight and hi-end: draft Russian encryption standard. In: Preproceedings CTCrypt 2014, June 05–06, 2014, Moscow, Russia. pp. 183–188, June 2014
62. Weiß, M., Heinz, B., Stumpf, F.: A cache timing attack on aes in virtualization environments. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 314–328. Springer, Heidelberg (2012)
63. Whiting, D., Housley, R., Ferguson, N.: Counter with CBC-MAC (CCM). IETF RFC 3610, September 2003. https://tools.ietf.org/html/rfc3610
64. Wu, H., Huang, T.: The Authenticated Cipher MORUS (v1). CAESAR First Round Submission, March 2014. http://competitions.cr.yp.to/round1/morusv1.pdf

A WHIRLBOB 1.0 π "8-bit" Reference Implementation

This ANSI C function implements the WHIRLBOB 512×512-bit π permutation.

```
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 come 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] ^= x;                         // 01
        pt[(j + 1) & 7] ^= x;                   // 01
        pt[(j + 3) & 7] ^= x;                   // 01
        pt[(j + 5) & 7] ^= x;
        pt[(j + 7) & 7] ^= x;

        // x <- 02
        x = (x << 1) ^ (x & 0x80 ? 0x1D : 0x00);
        pt[(j + 6) & 7] ^= x;                   // 02

        // x <- 04
        x = (x << 1) ^ (x & 0x80 ? 0x1D : 0x00);
        pt[(j + 2) & 7] ^= x;                   // 04
        pt[(j + 5) & 7] ^= x;                   // 01 + 04 = 05

        // x <- 08
        x = (x << 1) ^ (x & 0x80 ? 0x1D : 0x00);
        pt[(j + 4) & 7] ^= x;                   // 08
        pt[(j + 7) & 7] ^= x;                   // 01 + 08 = 09
      }
    }
  }
}
```