Introducing two Low-Latency Cipher Families: Sonic and SuperSonic

Yanis Belkheyar[1], Joan Daemen[1], Christoph Dobraunig[2], Santosh Ghosh[2] and Shahram Rasoolzadeh[1]

[1] Digital Security Group, Radboud University, Nijmegen, The Netherlands
firstname.lastname@ru.nl
[2] Intel Labs, Hillsboro, USA firstname.lastname@intel.com

Abstract. For many latency-critical operations in computer systems, like memory reads/writes, adding encryption can have a big impact on the performance. Hence, the existence of cryptographic primitives with good security properties and minimal latency is a key element in the wide-spread implementation of such security measures. In this paper, we introduce two new families of low-latency permutations/block ciphers called Sonic and SuperSonic, inspired by the Simon block ciphers.

Keywords: low-latency, Simon, Sonic, SuperSonic, Feistel structure, gate-delay-balenced Feistel, block cipher

1 Introduction

Low-latency ciphers are designed to encrypt and decrypt data with minimal delay, making them suited for pointer encoding or memory encryption where the latency is a critical element for the efficiency of a system. Their design is an interesting challenge because it involves finding a different trade-off than the usual one in cryptography: between security and total cost of computation. In low-latency ciphers the trade-off is between security and the latency of an optimized hardware implementation. It is determined by the *critical path* of the circuit, its worst-case gate delay. This alternative trade-off can result in unusual designs.

In this mindset we look at an existing design, the family of block ciphers Simon [BSS+13]. The distinguishing feature of Simon is its lightweight Feistel round function, that can be implemented in a circuit with a short critical path. Still, this does not translate to low latency for Simon as a high number of rounds are required. Therefore, we investigated ways to strengthen the round function without increasing its critical path. To do so, we introduce a variant of the Feistel construction that we call *gate-delay-balanced Feistel*. In this construction all paths through the round function have similar latency.

We introduce two cipher families that have the gate-delay-balanced Feistel structure called Sonic and SuperSonic. The constructions lend themselves to serve as cryptographic permutations or the datapath in block ciphers or tweakable block ciphers, where the round function takes as parameter a round constant in the case of a permutation and round key in the case of a (tweakable) block cipher. As opposed to the Simon block ciphers, for the members of our cipher families the inverse is not necessarily light. This is due to the fact that the inverse round function features the inverse of a mapping that we added to one of the Feistel branches. The parameters of this mapping can be chosen such that its inverse is light, e.g., it is an involution. With other choices implementing its inverse will have a high cost in number of operations and gate delay. However, such choices are advantageous for the differential and linear propagation properties for the round function

Table 1: Parameters for each Sonic v0.1 and SuperSonic v0.1 ciphers.

cipher	word size (w) [bits]	block size (b) [bits]	number of rounds	target security [bits]
Sonic256	128	256	24	128
Sonic512	256	512	24	128
SuperSonic256	128	256	21	128
SuperSonic512	256	512	21	128

and less rounds are required for a given target security strength. So there is a trade-off between the latencies of the forward and inverse computation.

The purpose of this note is to introduce our new design idea and we limit ourselves to permutations that we denote as Sonic v0.1 and SuperSonic v0.1. Both families have two members: one permutation with width 256 bits and one with width 512 bits. For all 4 permutations we make a 128-bit security claim, when used in the Even-Mansour [EM91, EM97] construction. We give an overview of the permutations introduced in this paper in Table 1.

We chose a version number of 0.1 for the following reasons. First, the choice of parameters in our permutation instances is based on a preliminary analysis. We plan to do a more in-depth analysis that will likely make us change their values. Second, we have chosen for a θ mapping with a heavy inverse, making the permutations proposed in this paper not well suited for modes of use that require the inverse. For some use cases like memory encryption that required fast read and can afford slower write heavy inverse is not a problem. We are planning to investigate different trade-offs with respect to those aspects. Third, our permutations are suited for use in other constructions such as sponge or duplex that may require fewer rounds and will certainly require additional security claims. Fourth, for smaller widths, (tweakable) block ciphers might be a better choice than permutations and we may extend the range of the families.

In the remainder of this note, we will introduce notation in Section 2, provide specifications for Sonic v0.1 and its design rationale in Section 3, do the same for SuperSonic v0.1 in Section 4 and finally give a formal security claim in Section 5.

2 Notation

In this paper we specify all processing as operating on w-bit words, that we denote by lowercase letters like x and y.

We denote the bit in position i in word x by x_i, where i ranges from 0 to $w-1$. We denote use the following notation for operations on words:

- $\overline{x}$: bitwise not (or complement)
- $x \,||\, y$: concatenation
- $x \oplus y$: bitwise addition
- $x \wedge y$: bitwise and
- $x \vee y$: bitwise or

Furthermore, we make use of the shift operation, cyclic shift operation, and multiplicative shuffle. We express the shift over an offset of t positions by σ^t:

$$y = \sigma^t(x): \quad y_i \leftarrow \begin{cases} x_{i+t} & \text{if } i < w - t\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

We express the cyclic shift over an offset of t positions by τ^t:

$$y = \tau^t(x): \quad y_i \leftarrow x_{(i+t) \bmod w} \ \forall i\,,$$

where it can be seen as t iterations of $\tau^1(\cdot)$ which we simply denote by $\tau(\cdot)$. We denote the multiplicative shuffle with factor s by π_s:

$$y = \pi_s(x): \quad y_i \leftarrow x_{(s\times i) \bmod w} \ \forall i\,.$$

3 Sonic

In this section, by specify the Sonic v0.1 cipher and provide a design rationale.

3.1 Specification of Sonic v0.1

Sonic is an iterated permutation with a balanced Feistel-like round function where processing is performed on both parts of the state. It applies 24 round functions and its block length is $2w$ while all operations are performed on w-bit words.

Round Function: The round function operates on two w-bit words denoted x and y and takes as parameter a round constant c^r:

$$x \,||\, y \leftarrow \pi_{15}\big(\gamma(x) \oplus \tau^{12}(x) \oplus y\big) \,||\, \theta(x) \oplus c^r\,,$$

where θ and γ are a bijective linear function and a nonlinear function respectively defined by

$$\theta(x) = \tau(x) \oplus \tau^8(x) \oplus \tau^{10}(x)\,,$$
$$\gamma(x) = x \wedge \tau(x)\,.$$

We provide a more formal specification of the Sonic v0.1 permutations in Algorithm 1 and compare the round function of Sonic with that of Simon in Figure 1.

Algorithm 1 Definition of Sonic v0.1

Parameters: number of rounds q
Input: $2w$-bit state s
Output: $2w$-bit state t
$x \,||\, y \leftarrow s$ with x and y both w bits long
for r from $1-q$ up to 0 **do** $(x, y) = \mathrm{R}[c^r](x, y)$
$t \leftarrow x \,||\, y$

The round function $\mathrm{R}[c^r]$:
$z \leftarrow x \wedge \tau(x)$ ▷ γ
$y \leftarrow y \oplus \tau^{12}(x) \oplus z$
$y \leftarrow \pi_{15}(y)$
$x \leftarrow \tau(x) \oplus \tau^8(x) \oplus \tau^{10}(x)$ ▷ θ
$x \leftarrow x \oplus c^r$
$(x, y) \leftarrow (y, x)$ ▷ swap

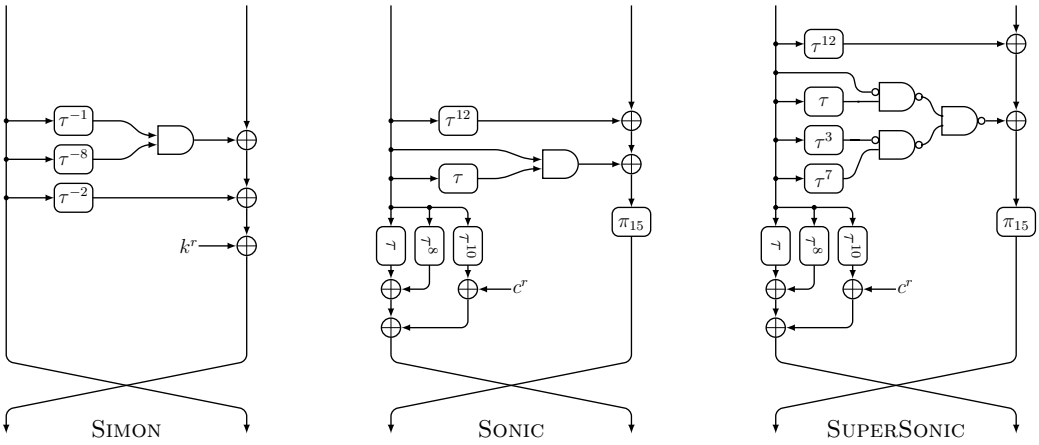

Figure 1: Comparison of round functions of SIMON, SONIC, and SUPERSONIC.

Table 2: The compact round constants e^r used in SONIC and SUPERSONIC ciphers.

r	-29	-28	-27	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17	-16	-15
e^r	01	20	0c	82	70	12	46	d9	0d	a2	7c	90	36	cb	4b
r	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0
e^r	7b	71	32	4a	5b	7d	b0	3a	49	3b	69	37	eb	47	f9

Round Constants: The round constants c^r of SONIC v0.1 are w-bit words and the bits with non-zero values are all in positions with index i a multiple of $w/8$. For each round constant c^r we can define an equivalent 8-bit *compact* constant e^r and their relation is

$$c_i \leftarrow \begin{cases} e_{\frac{8}{w}\times i} & \text{if } i \text{ divides } \frac{w}{8}\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

The round constants are computed recursively, with e^{r+1} computed from e^r applying the linear mapping:

$$e^{r+1} \leftarrow \tau^3(e^r) \oplus \sigma^2(e^r)\,.$$

We specify the compact round constants in Table 2.

For the numbering of the rounds we fix the number of the last round to 0 and count backwards. In the nominal case SONIC v0.1 has 24 rounds and they are numbered $-23, -22, \ldots, 0$. The result is that instances of SONIC v0.1 with different number of rounds will have the same round constant in the last round but different ones in the first round. For instance, for full-round SONIC, the first and the last used round constant values are `46` and `f9`, respectively.

3.2 Design Rationale

Consider the structure of a classical Feistel round function, as depicted in Figure 2. In this round function, the critical path is in the datapath from the input to the left output word through the f function. Clearly, the combinatorial circuit for this function can be optimized to minimize the critical path. That is the reason for using an alternative representation of the f function in Figure 2. It takes as input both left and right words

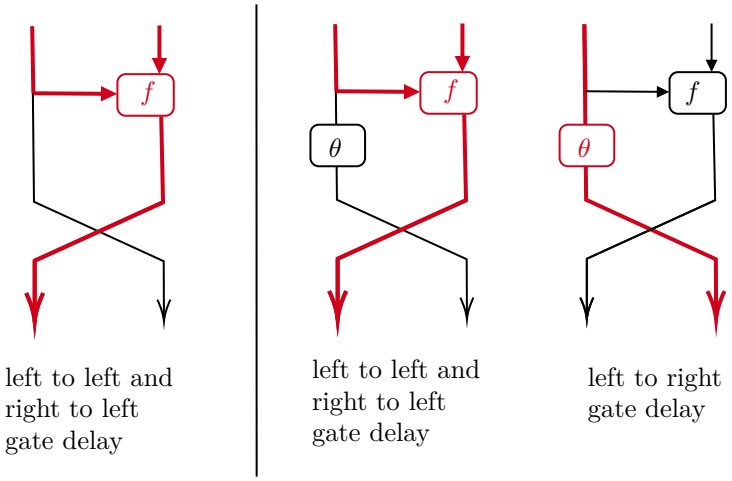

Figure 2: Possible critical paths in a Feistel (left) and in a gate-delay-balanced Feistel round functions (right).

instead of the classical representation that has f operating on the left word and adding its output to the right word.

In a Feistel round function, the right word of the output is simply a copy of the left word of the input. We can add processing in this branch increasing the strength of the round function with limited impact on its critical path. Increased strength in the round function in turn allows reducing the number of rounds for the same security margin. In particular, we insert in the branch a linear mixing layer θ, as shown in Figure 2. By choosing a mixing layer with gate depth smaller than that of the f function, the increase in gate delay of the round function will be very small.

When we remove the round key addition, the round function of Simon requires two 2-bit `XOR` gates and one 2-bit `AND` gate for each bit in the left output word. As illustrated in Figure 1 this can be implemented with a gate depth of two 2-bit `XOR` gates. For θ we chose the well vetted diffusion block of adding three rotated words [DGV91, NIS12, DEMS21, DMMR20, BDD+23]. A similar concept has been used concurrently in Arion [IIL+23] to modify Simpira [GM16], although there the function added in the empty branch is non-linear.

Our θ mapping can be implemented using two 2-bit `XOR` gates per output bit and with a gate depth of two 2-bit `XOR` gates. Therefore, the overall gate depth of the Sonic round function is two 2-bit `XOR` gates. Moreover, by adding a third `XOR` gate we can include the addition of a round constant (or round key) without increasing the gate depth.

A second modification as compared to the Simon round function is the inclusion of a multiplicative shuffle, denoted as π in the specification. The concept of multiplicative shuffle was introduced in [DGV91] and is also used in Subterranean2.0 [DMMR20]. In terms of gate delay, a bit shuffle has negligible cost as it is just wiring. We added the multiplicative shuffle to improve the increase of degree of the algebraic normal form expressions of output bits in terms of input bits for reduced-round versions of Sonic v0.1.

4 SuperSonic

In this section, we specify the SuperSonic v0.1 permutations that can be seen as an enhancement of Sonic and provide a design rationale.

4.1 Specification of SuperSonic v0.1

The round function and round constants of SUPERSONIC match those of the SONIC, except that it applies a different non-linear function γ:

$$\gamma(x) = \left(\overline{x} \wedge \tau(x)\right) \vee \left(\tau^3(\overline{x}) \wedge \tau^7(x)\right).$$

For a comparison with SONIC and SIMON, we refer to Figure 1.

Another difference between SUPERSONIC v0.1 and SONIC v0.1 is the number of rounds: while SONIC has 24 rounds, SUPERSONIC has only 21.

The round constant values and numbering in SUPERSONIC v0.1 is the same as in SONIC v0.1. For instance, for full-round SUPERSONIC, the first and the last used round constants value are `a2` and `f9`, respectively.

4.2 Design Rationale

As it can be seen in Figure 1, in SONIC there are two different types of path from the output word to the input: several ones with gate depth two 2-bit `XOR` gates, and one with one 2-bit `AND` and one 2-bit `XOR` gate. In most technologies a 2-bit `AND` has a lower gate delay than 2-bit `XOR`. Therefore, we can replace the 2-bit `AND` gate corresponding to the γ function by a more complex circuit with only limited impact on the critical path.

In SUPERSONIC we replace the 2-bit `AND` gate by two "stacked" 2-bit `NAND` gates. More precisely, the output word of γ in SUPERSONIC is the bitwise `OR` of two intermediate words. Each of these intermediate words is in turn the bitwise `AND` of two shifted versions of the input word, one of them complemented. By using De Morgan's laws we can map this readily to a two-layer `NAND` circuit (or to a compact `AOI22` gate), known for its small gate delay. We added the complement operations based on preliminary analysis and this analysis shows an increase of algebraic degree of the round function from 2 to 4 and an improvement of the differential and linear propagation properties. This allows us to reduce the number of rounds compared to SONIC v0.1.

5 Security Claim

For all four permutations in Table 1, we make a security claim for the block cipher that we obtain by using the permutation in the Even-Mansour construction [EM97], with a $2w$-bit secret key K.

Claim. *For a given permutation* Perm *we define the block cipher* $\mathrm{BC}[\mathrm{Perm}]_K$ *by*

$$\mathrm{BC}[\mathrm{Perm}]_K(X) = \mathrm{Perm}(X \oplus K) \oplus K\,,$$

with K a secret key of the same length as the width of the permutation Perm*. For all* $\mathrm{Perm} \in \{\textit{Sonic256}, \textit{Sonic512}, \textit{SuperSonic256}, \textit{SuperSonic512}\}$*, the advantage of distinguishing* $\mathrm{BC}[\mathrm{Perm}]_K$ *with a uniformly chosen secret key from a random permutation is upper bounded by*

$$\frac{N+M}{2^{128}}\,,$$

where N is the computational complexity expressed in number of computations of Perm *(or its inverse) and M is the data complexity in number of queries to* $\mathrm{BC}[\mathrm{Perm}]_K$ *(or its inverse).*

6 Implementation Aspects

In the classical Feistel the inverse cipher/permutation simply consists of the same sequence of operations but with the order of the round constants/keys reversed. However, the introduction of θ destroys this property: for computing the inverse of the permutation, the inverse of θ must be implemented.

The θ mapping by itself is lightweight in that each output bit is the sum of only 3 input bits. Its inverse however is rather heavyweight: For $w = 128$ and $w = 256$, each output bit of θ^{-1} is the sum of 65 and 129 input bits respectively. Hence, implementing θ^{-1} necessitates a gate depth of at least seven and eight 2-bit `XOR` gates respectively. Furthermore, the high fan-out number of input bits significantly increases the gate delay of the inverse round function circuit. As a result, for modes of use that require both low-latency forward and inverse execution of the permutation, Sonic v0.1 and SuperSonic v0.1 are less suited. However, variants can be specified with a different shift offsets in θ with a much lighter inverse, e.g., in the extreme case an involutive θ.

Both Sonic and SuperSonic inherit the lightweight round function from Simon. This lightweight characteristic is not limited to the gate delay of the function but also extends to the footprint area of the implementation. The Sonic round function can be implemented using one 2-bit `NAND` gate and four 2-bit `XOR`/`XNOR` gates for each w bit. On the other hand, the SuperSonic round function can be implemented using an `AOI22` gate and four 2-bit `XOR`/`XNOR` gates. It is important to note that these gate counts refer to the combinatorial circuit of the round functions themselves. To implement the entire cipher, additional layers of `BUF`/`INV` gates are needed to connect the consecutive round functions.

To ensure low-latency performance, our primary objective is to have excellent performance in a fully unrolled implementation of Sonic and SuperSonic in ASIC hardware technology. This means that the entire cipher is implemented in a single combinatorial circuit without any intermediate register stages. However, to connect the sub-circuits corresponding to each pair of consecutive round functions, a layer of `BUF`/`INV` gates is applied to serialize the data flow. Our preliminary synthesis results indicate that both Sonic and SuperSonic exhibit a latency significantly lower than that of Simon-128, the largest member of Simon family.

Despite having the same gate depth as unkeyed latency-optimized Simon round function implementation, the implementation of the Sonic and SuperSonic round functions are likely to have a larger critical path. As a matter of fact, adding θ increases the fan-out of the bits of the right input word from 4 to 6 and this increases the critical path. Moreover, applying the stronger γ function in SuperSonic, further increases this number from 6 to 8. However, by applying certain techniques explained in [LMMR21], this increase can be limited.

In spite of the focus on suitability for low-latency hardware, Sonic performs quite well in software. Processing in Sonic consists exclusively of three types of operations: word-wise Boolean operations, cyclic shifts and the multiplicative shuffle. The Boolean operations can be implemented efficiently with bitwise Boolean instructions. Cyclic shifts of 128 and 256-bit words can be implemented efficiently on CPUs with word lengths of 32 or 64 bit by using bit interleaving techniques [BDP+12]. As for the multiplicative shuffle, in [DMMR20] it is described how to avoid computing it altogether by adopting a technique called π *procrastination*.

References

[BDD+23] Yanis Belkheyar, Joan Daemen, Christoph Dobraunig, Santosh Ghosh, and Shahram Rasoolzadeh. Bipbip: A low-latency tweakable block cipher with

small dimensions. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2023(1):326–368, 2023.

[BDP+12] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer. KECCAK implementation overview, May 2012. https://keccak.team/papers.html.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. IACR Cryptol. ePrint Arch., page 404, 2013.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. J. Cryptol., 34(3):33, 2021.

[DGV91] Joan Daemen, René Govaerts, and Joos Vandewalle. A framework for the design of one-way hash functions including cryptanalysis of damgård's one-way function based on a cellular automaton. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, Advances in Cryptology - ASIACRYPT '91, International Conference on the Theory and Applications of Cryptology, Fujiyoshida, Japan, November 11-14, 1991, Proceedings, volume 739 of Lecture Notes in Computer Science, pages 82–96. Springer, 1991.

[DMMR20] Joan Daemen, Pedro Maat Costa Massolino, Alireza Mehrdad, and Yann Rotella. The subterranean 2.0 cipher suite. IACR Trans. Symmetric Cryptol., 2020(S1):262–294, 2020.

[EM91] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, Advances in Cryptology - ASIACRYPT '91, International Conference on the Theory and Applications of Cryptology, Fujiyoshida, Japan, November 11-14, 1991, Proceedings, volume 739 of Lecture Notes in Computer Science, pages 210–224. Springer, 1991.

[EM97] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. J. Cryptol., 10(3):151–162, 1997.

[GM16] Shay Gueron and Nicky Mouha. Simpira v2: A family of efficient permutations using the AES round function. In Jung Hee Cheon and Tsuyoshi Takagi, editors, Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I, volume 10031 of Lecture Notes in Computer Science, pages 95–125, 2016.

[IIL+23] Takanori Isobe, Ryoma Ito, Fukang Liu, Kazuhiko Minematsu, Motoki Nakahashi, Kosei Sakamoto, and Rentaro Shiba. Areion: Highly-efficient permutations and its applications to hash functions for short input. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2023(2):115–154, 2023.

[LMMR21] Gregor Leander, Thorben Moos, Amir Moradi, and Shahram Rasoolzadeh. The SPEEDY family of block ciphers engineering an ultra low-latency cipher from gate level for secure processor architectures. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2021(4):510–545, 2021.

[NIS12] NIST. FIPS PUB 180-4: Secure hash standard (shs), 2012-03-06 2012.