Hardware Bit-Mixers

Laszlo Hars
January, 2016

Abstract—A new concept, the *Bit-Mixer* is introduced. It is a function of fixed, possibly different size of input and output, which computes statistically uncorrelated output from correlated input values, and its behavior is altered by parameters, called keys. Several constructions are presented, with very fast, power efficient implementations in electronic hardware, having very little side channel leakage. In information security bit-mixers have many applications, mostly when their output is hidden from an adversary. They include key generators, parallel stream ciphers, hash functions, data dependent authentication codes, and many more.

Keywords—Information security, cryptography, cryptographic hardware, electronics, side channel analysis, side channel attack

I. Introduction

Many information security applications require high-performance, fix-sized input and output functions which thoroughly “mix” their input value. These functions, which are called bit-mixers, produce statistically uncorrelated output from correlated input values. E.g. any “simple” change in the input causes on average half of the out-put bits to change. Bit-mixers also utilize parameters (keys), which make their behavior unpredictable to an observer.

While performance and power consumption are critical in embedded applications, advanced VLSI technologies provide designers the ability to improve security by modest increase of circuit size. The extra gates add normally very little to the costs.

Even though many other uses are feasible, the most important applications are in which one or both the input and output interfaces are internal to the design and thus hidden from the observer. In these instances the cryptographic requirements beyond a generalized strict avalanche criterion are minimized if not eliminated. Specifically, the primary remaining attacks exploit data-dependent information exposed through the circuit’s side channel emanations, including variations in response time, electromagnetic radiation, fluctuations in power consumption…

We define bit-mixers as mathematical functions, propose definitions for good mixing properties, describe and analyze three sets of constructions of thorough, arbitrary size bit-mixers which are high-performance, low-power and minimize their side channel leakage, when implemented in electronic hardware. We laid out each construction in a 32 nm Silicon on Insulator (SOI) ASIC and verified these claims with a Differential Power Analysis (DPA) workstation.

In [1] many information security applications of bit-mixers are discussed. We invite further research on constructing bit-mixers and analyzing their use in security applications.

II. Bit-Mixers

One may generally think of bit-mixing as performed by reduced round ciphers with arbitrary block sizes. The input can be padded, or the output folded together (via XOR functions or compression S-Boxes) for the required sizes of the input and output of the bit-mixer. While there are indeed other constructions, the desired properties of bit-mixers are:

1. The fixed lengths of the input and output values can be independently and arbitrarily chosen
2. Every input bit affects every output bit
3. *Simple changes* in the input cause on average half of the output bits to change
4. A series of *simple changes* in the input yield output values without apparent correlation to the input or to the change pattern of the input, i.e. standard statistical tests accept the output sequence as random
5. Parameters (keys) alter the behavior of the function

As the term “*simple change*” above can have various definitions, further research will ultimately determine which definition proves to be the most suitable. More pointedly, we may find that a simple change is one in which less than half of the bits change simultaneously or even one which results from a software-based transformation using fewer than a dozen instructions.

It is instructive to note that property 3 above is a generalization of the Strict Avalanche Criterion [2].

III. Mixing Quality

Besides theoretical considerations, we employed test methodologies similar to differential cryptanalysis to verify good mixing properties. For iterative bit-mixers, the number of rounds was determined, required to generate output values satisfying the Strict Avalanche Criterion [2], i.e. are statistically random, after changing single input bits. For illustration purposes, graphic representations of data evolutions are also plotted.

The bitwise differences of the individual rounds of iterative bit-mixers were tested using the following 3σ statistics:

- The Hamming weight
- The number of times any bit pair occurs
- The number of times any bit triplet occurs
- The longest run of equal consecutive bits
- The length of the 2nd, 3rd, … 10th longest runs of equal bits

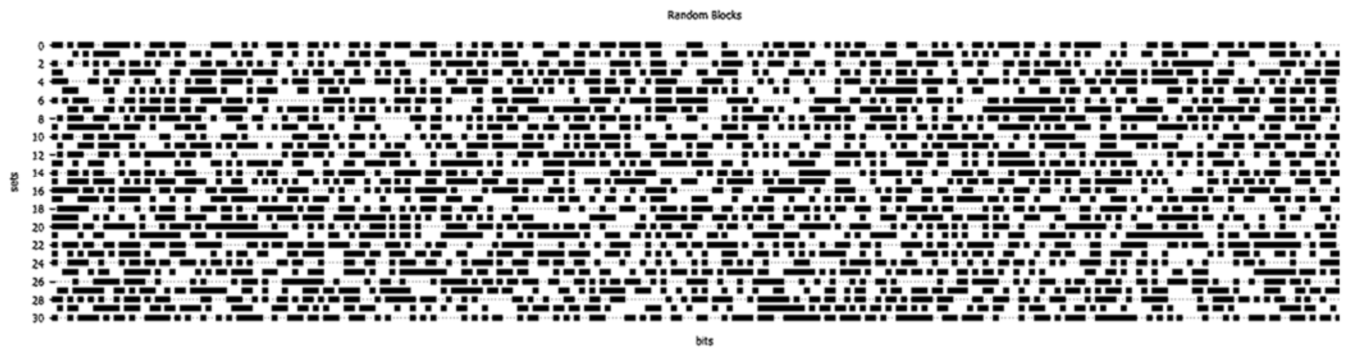

Figure 2. True random rows of dots

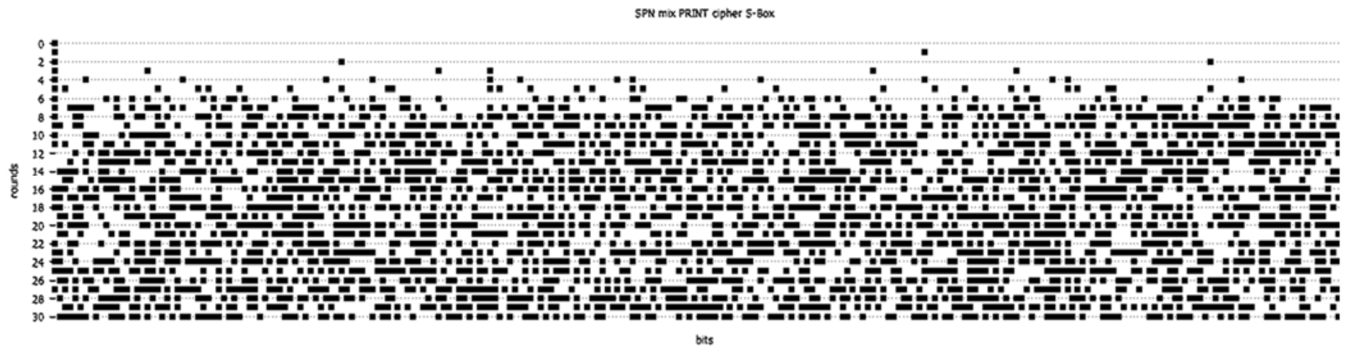

Figure 1. Differential behavior of an iterative bit-mixer

When random subkeys are used for testing, we can set the input to all 0's. 1000 or more unrelated keys were tried as certain intermediate bits may unpredictably interact with keys.

The results of a typical experiment are plotted in Figure 1 with rows of dots corresponding to rounds, iterations of mixing steps. The dots show the positions of the output bits which changed in that round, caused by a flip of a single input bit.

When compared to a plot of true random numbers on Figure 2, the results in row 8 and beyond "look" equally random. It confirmed visually the outcome of our statistical tests.

To verify statistical randomness (property 4) data sets of 10 million bytes were generated from input values 0, 1, 2… and tested with Diehard- [17] and with the NIST statistical randomness tests [18]. Because of the regular structure of the presented bit-mixer constructions, shifted or permuted versions of input sequences behave similarly.

IV. XOR-Tree based Bit-Mixers

In XOR-tree based bit-mixers, the input is partitioned into multiple, possibly different length bit groups. Using multiplexers the bits of the groups select certain parts of the key material, called subkeys. These subkeys are bitwise XOR-ed together to generate the final bit-mixer output as shown in Figure 3.

While the XOR operation in ASICs is typically implemented using a tree of 2-input XOR gates, multi-input gates or parity generators can be used depending on the target technology, e.g. in FPGAs that provide wide lookup tables.

Bit-mixers of this construction are straightforward to implement and offer high performance, improved security, low power consumption, and a minimal side channel leakage.

A. *Properties of the XOR-tree Construction*

As the width of the input and the width of the output of XOR-tree based bit-mixers can be independently chosen, expansion and compression functions are created by selecting a longer output width or longer input width, respectively.

Having random key material, any single input bit change will cause the output to change by a random subkey. As such every output bit is influenced by any input bit change. Further, given the bit-mixers construction, multiple input bit changes will cause the output to change by an XOR-ed aggregation of random subkeys which is in itself random. Therefore, XOR-tree based bit-mixers satisfy each of the desired properties as enumerated above and ensure theoretically perfect mixing.

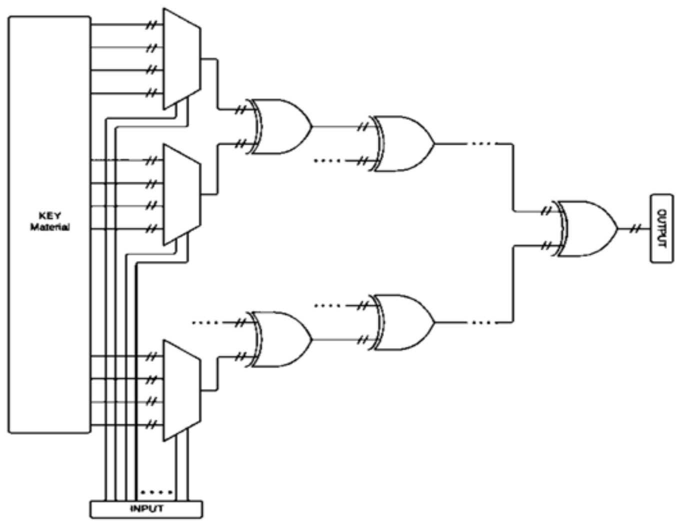

Figure 3. XOR-tree Bit-Mixer

B. *Performance of the XOR-Tree Construction*

The circuit we evaluated in our test ASIC expanded an 80-bit input into a 256-bit output utilizing 2-to-1 multiplexers for subkey selection and 2-input XOR gates to implement a seven layer XOR-tree. Even with the limited fanout/loading of the gates within the circuit, the bit-mixer circuit can operate in one clock cycle in systems with clock rates in excess of 1.2 GHz.

C. *4-Way Correlation – Linearity*

The XOR-tree construction is linear in a binary Galois field, because it only uses bit selection and XOR operations. In these constructions some 4-way correlations exist among the output values computed from simply correlated input values. In this instance, correlations arise as follows:

Assuming at least 2*-bit input groups, choose a bit* b *from one of the input bit-groups* B*, and bit* c *from a different input bit-group* C*. Holding all bits of group* B *except* b *constant, let* K0 *denote the subkey selected when* b *is logic* 0 *and* K1 *denote the subkey selected with* b *is logic* 1*. Similarly, let* L0 *and* L1 *denote the subkeys selected based on the logical value of* c *while other bits of group* C *are held constant. Finally, let* M *denote the XOR of all subkeys selected by other input bit-groups where their inputs are held constant (*M = 0…0 *if there is no more bit-group). The bitwise XOR of the output values resulted from all possible 2x2 values of* b *and* c *will yield a vector of* 0*'s, what we call "4-way correlation".*

$$(M \oplus K0 \oplus L0) \oplus (M \oplus K1 \oplus L0) \oplus$$
$$(M \oplus K0 \oplus L1) \oplus (M \oplus K1 \oplus L1) = 0\ldots0$$

In applications where the output values cannot be observed, this type of correlation does not pose security problems. For applications where this correlation *is* a concern, the output can be further processed by a nonlinear function such as:

- A parallel collection of nonlinear functions such as S-Boxes. See in [3], [4] and [5].
- The outputs of a collection of nonlinear functions such as S-Boxes, XOR-ed with the original output: [6], [7].
- Rotate-Add-XOR (RAX) constructions as described in [8] and [9] (suitable for microprocessor implementations)

Another way to make the construction nonlinear, is to replace the XOR operations in one or more levels of the XOR-tree with nonlinear compressing S-Boxes, similar to the one shown in Figure 5. While straightforward to implement, the resulting uneven circuit delays may require manual balancing for low side channel leakage. With moderate effort, replacing two layers of the XOR-tree with 4-to-1 S-Boxes achieves single clock cycle operation still at clock rates upwards of 1.2 GHz.

V. Substitution-Permutation Network based Bit-Mixers

Invertible bit-mixers can be defined based on the well-known substitution-permutation networks, SPNs [10]. For compression or expansion bit-mixers the block size $\mathcal{B}$ of the SPN is chosen to be the larger of the input $\mathcal{I}$ and output $\mathcal{O}$ sizes of the bit-mixer. If $\mathcal{I} < \mathcal{B}$, keep the unused bits of $\mathcal{B}$ constant or repeat some bits. If $\mathcal{O} < \mathcal{B}$, discard bits of $\mathcal{B}$ or fold bits of $\mathcal{B}$ together via XOR or S-Box functions to produce the output of the bit-mixer.

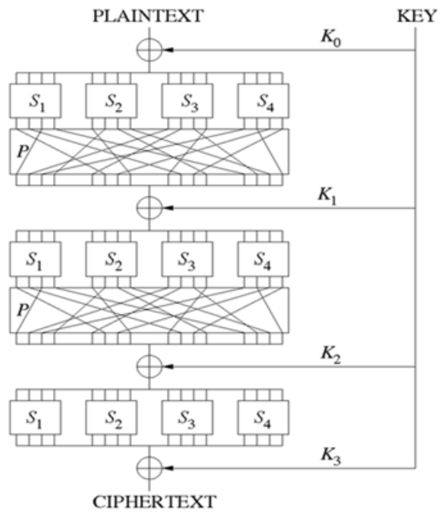

Figure 4. Substitution-Permutation Network

Substitution-Permutation (SP) networks are customarily built according to Figure 4 with iterations of the following 3 steps:

1. The input is transformed by a series of nonlinear functions, S-Boxes
2. The bits of the result are rerouted, permuted
3. The permuted data is XOR-ed with a round key (in bit-mixers it is called "subkey")

Note that the first and last round are often simplified, omitting one or two steps.

If the S-Boxes are invertible, the SP Network will also be invertible, and if the S-Boxes are nonlinear, the SP network will be nonlinear, as well.

SP networks can be arbitrarily wide but the number of rounds required for a thorough mixing depends on this width, as discussed below in section C.

A. *S-Boxes*

There are many S-Boxes described in literature which are appropriate for use in SP networks, e.g. in [5]. In hardware implementations, small S-Boxes yield faster bit-mixers. The smallest practical S-Box, one with 3 input bits and 3 output bits, is implemented in PRINTcipher [3]. The three output bits of this 3x3 S-Box are defined as follows:

```
F0 = A  B' C' + A' (C  + B )
F1 = A' B  C  + B' (C' + A )
F2 = A  B  C' + C  (B' + A')
```

We designed small and fast circuits to implement this S-Box. They require only a handful of gates for each output bit, shown in Figure 6.

Similarly, the PRESENT cipher [4] uses the 4x4 S-Boxes as follows:

```
F0 = A'B C' + A C D + A'B D'   + A B'C' + A C'D'
F1 = A'B'C  + B C'D + A B'C'D' + A B C  + B C D'
F2 = A B C' + A C'D + A'B'D'   + A'B C  + A'B D
F3 =A'B D+A'C'D+A B D'+A C'D'+A B'C D+A'B'C D'
```

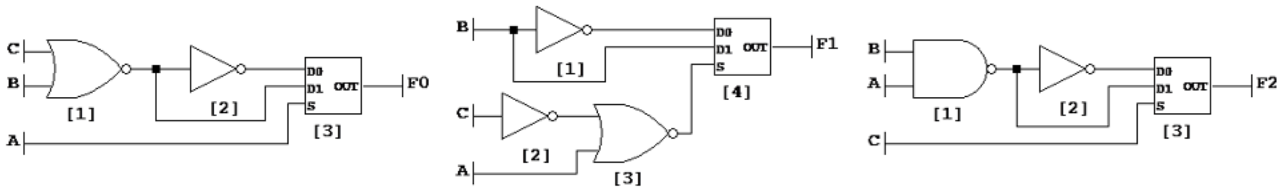

Figure 6. PRINTcipher S-Box circuits

Our circuit designs to implement this S-Box required twice as many gates as the PRINTcipher. See e.g. F1 in Figure 5.

Many other good 4x4 S-Boxes are discussed in [5]. They offer similar performance and mixing properties. Simpler, faster S-Boxes can also be used, although they require additional rounds to achieve the same thorough mixing properties, which effectively reduces the overall performance of the bit-mixer.

B. Permutation

Many suitable permutations have been published for ciphers such as PRINTcipher, the ciphers PRESENT and AES [10] as well as for hash functions such as SHA3 [11]. The simple permutation used in the first two ciphers above achieves perfect dispersion in the first few rounds; the bits affected by a single input bit-flip are fed into different S-boxes. This permutation, where the input block size to be mixed is b and the width of the S-Box is s, is defined as follows:

$$\mathrm{P}(i) = s \cdot i \bmod b-1 \;\text{ for }\; 0 \le i \le b-2;\text{ and } \mathrm{P}(b-1) = b-1$$

C. Number of Layers (Rounds)

A b-by-b S-Box distributes a single input bit-flip to b bits of the next round. A proper permutation routes these bits to different S-Boxes of the next round, distributing the changes to b^2 bits. After r rounds, a single bit-flip in the input affects b^r output bits until all bits are affected. We want $b^r \ge n$, that is a single input bit affects all output bits: $r \ge \log(n) / \log(b)$. Naturally, more rounds will achieve more thorough mixing.

D. Mixing Properties with the PRINTcipher S-Box

In our implementation of an SP network using PRINTcipher S-Boxes, the block size was 255 bits. For a perfect mixing, the minimum number of rounds required is $\log(255) / \log(3) \approx 5$. A few cases from 1000 random key sets required more rounds, but 9 rounds always achieved statistically perfect mixing. Figure 1 shows typical improvements of mixing with the rounds, which “look” perfect already after 8 rounds. Executing 9 rounds in a single clock cycle, as needed in the worst cases, allows clock rates upwards of 500 MHz.

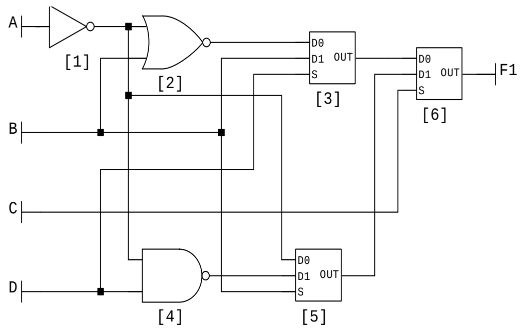

Figure 5. F1 output of the PRESENT cipher S-Box

E. Mixing Properties with the PRESENT Cipher S-Box

Using the PRESENT cipher S-Boxes in our implementation of another SP network, we set the input and output width to 256 bits. To achieve perfect mixing, the minimum number of rounds required is $\log(256) / \log(4) = 4$, but a few of our statistical tests of 1000 random key sets required 6 rounds, to achieve perfect mixing. Figure 7 shows typical mixing properties, which “look” perfect after 5 rounds. Even at a worst case 6 rounds, SP networks utilizing the PRESENT cipher S-Box require 3 fewer rounds than those that utilize the PRINTcipher S-Box, as they mix in each round more thoroughly, farther from any linear function. The difference in the number of rounds yields a performance increase. Executing all 6 rounds in a single clock cycle allows clock rates upwards of 600 MHz.

VI. Double-Mix Feistel Network based Bit-Mixers

We devised another family of invertible bit-mixers based on a new type of mixing operation, a balanced variant of Feistel Networks [7]. Similar to the SP network based bit-mixers, the block size can be the larger of the input and the output size, repeating input bits or folding output bits as required for compressing or expanding bit-mixers.

Even though Feistel ciphers [7] transform only half their input bits in each round, direct implementation in software can completely consume a CPU. On the other hand, parallel

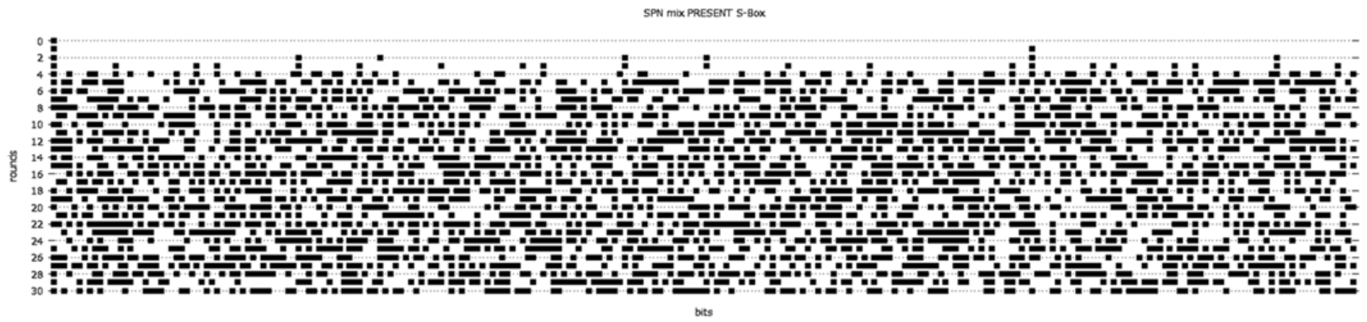

Figure 7. Differential behavior of an SP Network Bit-Mixer with the PRESENT Cipher S-Box

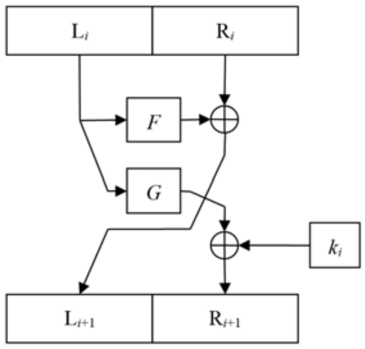

Figure 8. One round of a Double-Mix Feistel Network

hardware implementations can gain a twofold speedup for the same mixing quality by transforming all intermediate data. It is achieved by a Feistel Network variant, which we call the Double-Mix Feistel Network (DMFN) as shown in Figure 8.

In DMFN the data is processed in rounds similar to Feistel Networks. The data is handled in two halves L and R. In each round, Round$_i$, two functions F and G compute values from L_i and R_i, which give L_{i+1}, R_{i+1} after 2 XOR operations. The very first values L_0 and R_0 are set to the input of the bit-mixer and the very last values L_r, R_r constitute the output.

While L_{i+1} is generated using a bitwise XOR operation of the output of F and R_i, a round key k_i is mixed-in using a bitwise XOR operation with the output of G to generate R_{i+1} as follows:

$$L_{i+1} = F(L_i) \oplus R_i$$
$$R_{i+1} = G(L_i) \oplus k_i$$

If we need invertible bit-mixers, G must to be an invertible function. The inverse of G need not be easily computed unless the application uses the inverse of the bit-mixer. As such G can be faster computable than a typical S-Box layer, and it can process bits in distant positions, mixing the data better.

An example of such a function is XOR-ing each input bit of G with two input bits from circular distances (d_1, d_2), taking minimal time in electronics. At power-of-two block lengths, these 3-way XORs define invertible functions, as proved in [9].

F does not have to be invertible as its inverse is not needed even for the inverse of the bit-mixer. In our implementations, we used a fast, sufficiently complex construction, which is nonlinear in the Galois field of binary polynomials, as follows:

1. NAND bits of L_i from circular distances d_3 and d_4
2. NOR bits of L_i at circular distances d_5 and d_6
3. NAND bits of L_i at circular distances d_7 and d_8
4. XOR the above three blocks of bits to L_i

In hardware implementations, shifts are wirings, consuming little time. F and G are nearly equal in path length requiring only a moderate amount of manual effort to balance the critical timing paths, needed for reduced side channel leakage. While F and G could be different in certain, if not all rounds, in our tests, for simplicity, we kept them the same in all rounds.

An invertible function G makes the DMFN invertible: one can compute from bottom up, i.e. from R_{i+1} compute L_i, knowing the round key k_i and the inverse of G. Having L_i compute $F(L_i)$, which is XOR-ed to L_{i+1} to yield R_i.

Invertibility can be useful for ensuring that all possible output values occur once, computed from certain unique input values.

As described previously, in each round only half-length subkeys (k_i) are mixed-in with G. We found no noticeable mixing improvements with subkeys of the full block length, realized e.g. if another half-length subkey was XORed to R_i.

A. Mixing Properties

In our DMFN implementation, the input and output width was 256 bits. Thousands of software simulation runs led to good sets of shift distances. For example, Figure 9 shows the evolution of mixing using the following shift distances:

$$\{d_1, d_2 \ldots, d_8\} = \{9, 73, 1, 17, 6, 25, 11, 26\}.$$

In 1000 tests using random key material, we found that 6 rounds were always enough to achieve statistically perfect mixing. Implementing all 6 rounds in a single clock cycle allows a clock rate upwards of 660 MHz.

VII. Side Channel Attack Resistance

Even though a function may be cryptographically secure, its physical implementation could leak information about the data and or keys via side channels. Relevant side channels include response time variations, fluctuations in power consumption, electromagnetic emanations, and even varying voltage levels on device pins. See, for example, [12] and [13].

Because the functions described above can be implemented in asynchronous circuits of simple combinatorial logic gates, side channel leakage is minimized. More pointedly, as the circuits do not require structures that are typically the main source of side channel leakage such as flip-flops, latches and

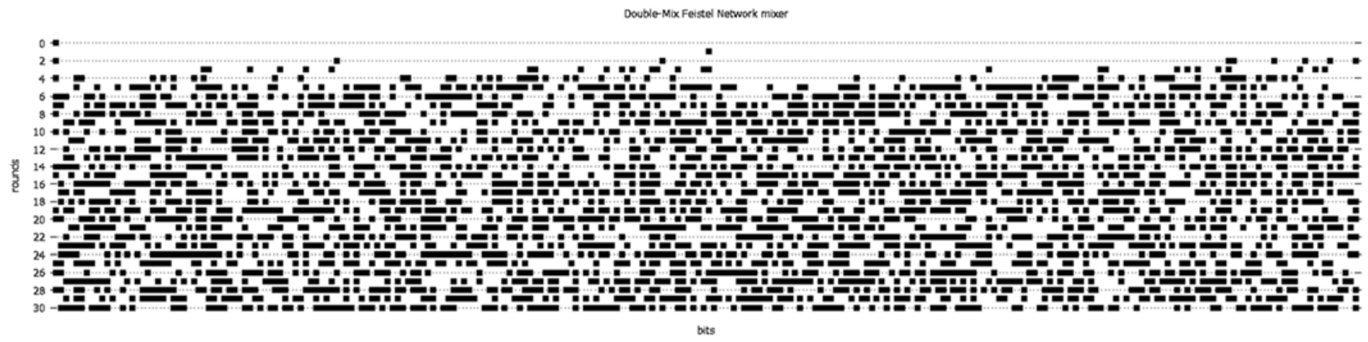

Figure 9. Differential behavior of a DMFN bit-mixer with shift distances: {9, 73, 1, 17, 6, 25, 11, 26}

other types of storage devices, the circuits are less susceptible to side channel analysis.

Variations of the lengths of the signal paths that may still exist can be reduced using manual layout techniques [14] to balance the already highly symmetric paths, thereby ensuring that many concurrent switching events occur at almost exactly the same time. This balancing step may not be necessary, because switching transients in e.g. our test ASIC's 32 nm SOI target technology are in the picoseconds. Recording/analyzing such transients in an effort to mount a template attack [15] would require a data acquisition system with a sampling rate in the THz range, an order of magnitude faster than available in the foreseeable future.

Using a DPA side channel analysis workstation [16], no exploitable side channel leakage was measured, such as correlations between power traces and output bits while varying the input bits. Note that other type of physical attacks have to be mitigated at the applications. They include probing [19] and fault injection [20].

VIII. Key Material

While different subkeys taken from the key material can share bits, there are obvious restrictions. E.g. for XOR-tree bit-mixers the same key material bit must not appear in the same position of multiple subkeys, as the XOR operations could effectively cancel this bit. That in mind, a simple bit reuse method is to generate a few subkeys by rotating a block of key material bits. Rotation as well as more complex mappings can be used to reduce the size of the key storage or minimize the bandwidth required to distribute keys.

Another solution for key distribution at limited bandwidth employs a second bit-mixer with hardcoded key material. From a shorter key the second bit-mixer can iteratively generate subkeys for the first bit-mixer. Ciphers and cryptographic hash functions can also be used to generate key material before use.

IX. Software Implementations

While the bit-mixers listed above were optimized for hardware implementation, they work well when implemented in software, too, even though other constructions are also viable. Software bit-mixing in single clock cycles is not possible, but bit-mixers can still operate orders of magnitude faster than ciphers or hash functions of similar input and output sizes. When no high security, only statistical independence of some generated data is required, one can save significant computation time even in software.

A family of bit-mixers is based on Rotate-Add-XOR (RAX) constructions. It is well suited for software implementations. Below is a 64-bit example, taken from [9]. The constants are hard coded subkeys, to be replaced with subkeys from the key storage. In the following pseudocode the function ROL is ROtate-Left and the internal variable k is initialized to 0:

```
x = (k += 0x3779884922721DEB)
x = (x^ROL(x,L)^ROL(x,R))+0x49A8D5B36969F969
x = (x^ROL(x,L)^ROL(x,R))+0x6969F96949A8D5B3
x = (x^ROL(x,L)^ROL(x,R)).
```

X. Summary

We introduced the concept of "bit-mixers", with possible alternative definitions and measures for the quality of mixing. Three families of example constructions were discussed, which are extremely fast with little side channel leakage. The input of the XOR-tree based bit-mixer constructions select sub-keys from a key storage, to be mixed together by bit-wise XOR operations. The second family of bit-mixer constructions uses the well-known substitution-permutation networks, presented with optimized implementations of small S-Boxes. The third group of bit-mixer constructions employs new circuits, called "double-mix Feistel networks", with appropriate component functions optimized by extensive simulations. The mixing quality of all the constructions were experimentally verified.

References

[1] Laszlo Hars "Information Security Applications of Bit-Mixers." Cryptology ePrint Archive 2017.

[2] Webster, A. F.; Tavares, Stafford E. "On the design of S-boxes". Advances in Cryptology - Crypto '85. Lecture Notes in Computer Science 218. New York, NY, Springer-Verlag New York, Inc. pp. 523–534.

[3] Lars Knudsen, Gregor Leander, Axel Poschmann, Matthew J. B. Robshaw. "PRINTcipher: A Block Cipher for IC-Printing." Cryptographic Hardware and Embedded Systems, CHES 2010 Volume 6225 of the series Lecture Notes in Computer Science, pp 16-32.

[4] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, C. Vikkelsoe. "PRESENT: An Ultra-Lightweight Block Cipher." Cryptographic Hardware and Embedded Systems - CHES 2007. Volume 4727 Lecture Notes in Computer Science pp 450-466

[5] Markku-Juhani O. Saarinen. "Cryptographic Analysis of All 4 × 4-Bit S-Boxes." Selected Areas in Cryptography. Volume 7118 Lecture Notes in Computer Science, pp 118-133.

[6] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. "The SIMON and SPECK lightweight block ciphers." In Proceedings of the 52nd Annual Design Automation Conference (DAC 2015). ACM, New York, NY, USA, Article 175 , 6 pages. DOI=http://dx.doi.org/10.1145/2744769.2747946

[7] D. Coppersmith. "The Data Encryption Standard (DES) and its Strength against Attacks." Technical report rc 186131994, IBM Thomas J. Watson Research Center, December 1994.

[8] L. Hars, G. Petruska: "Pseudorandom Recursions - Small and Fast Pseudorandom Number Generators for Embedded Applications." EURASIP Journal on Embedded Systems, vol. 2007, Article ID 98417, 13 pages, 2007. doi:10.1155/2007/98417.

[9] L. Hars, G. Petruska: "Pseudorandom Recursions II." EURASIP Journal on Embedded Systems 2012, 2012:1 doi:10.1186/1687-3963-2012-1.

[10] Kam, John B., and George I. Davida. "Structured design of substitution-permutation encryption networks." Computers, IEEE Transactions on 100.10 (1979): 747-753.

[11] Guido Bertoni, Joan Daemen1, Michael Peeters and Gilles Van Assche. "Keccak specifications." October 27, 2008. http://keccak.noekeon.org/

[12] Kocher, Paul. "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems". Advances in Cryptology—CRYPTO'96(1996). Lecture Notes in Computer Science 1109: 104–113.

[13] Kocher, Paul, Joshua Jaffe, and Benjamin Jun. "Differential power analysis." CRYPTO'99. Springer Berlin Heidelberg, 1999.

[14] Tiri, K., Verbauwhede, I.: "A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation." In: DATE '04: Proceedings of the conference on Design, automation and test in Europe, Washington, DC, USA, IEEE Computer Society (2004) 246–251

[15] Chari, Suresh, Josyula R. Rao, and Pankaj Rohatgi. "Template attacks." Cryptographic Hardware and Embedded Systems-CHES 2002. Springer Berlin Heidelberg, 2002. 13-28.

[16] Rambus: “DPA Workstation Analysis Platform.” https://www.rambus.com/security/dpa-countermeasures/dpa-workstation-platform/

[17] Marsaglia, George. "DIEHARD: a battery of tests of randomness." http://stat. fsu. edu/~ geo/diehard. html (1996).

[18] Rukhin, Andrew, et al. “A statistical test suite for random and pseudorandom number generators for cryptographic applications.” Booz-Allen and Hamilton Inc Mclean Va, 2001.

[19] Nikawa, K. "Applications of focused ion beam technique to failure analysis of very large scale integrations: A review." Journal of Vacuum Science & Technology B 9.5 (1991): 2566-2577.

[20] Hayashi, Yu-ichi, et al. "Non-invasive EMI-based fault injection attack against cryptographic modules." Electromagnetic Compatibility (EMC), 2011 IEEE International Symposium on. IEEE, 2011.