# Information Security Applications of Bit-Mixers

Laszlo Hars
January, 2016

**Abstract—A *Bit-Mixer* is a function of fixed size input and output, which computes uncorrelated output from correlated input values, and its behavior is altered by parameters, called keys. Several bit-mixer constructions have been published with very fast, power efficient implementations in electronic hardware, having very little side channel leakage. In this paper a dozen cryptographic applications are discussed, in most of which the output of the employed bit-mixers are hidden from an adversary. In these cases bit-mixers don't have to satisfy strict cryptographic requirements, but the security of the applications is improved by reducing exploitable correlations among intermediate values, and by diminishing side channel leakage of electronic implementations.**

***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 V, 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 output bits to change. Bit-mixers may also utilize secret keys, so their behavior becomes unpredictable to an observer.

While performance and power consumption are critical in embedded applications, advanced VLSI technologies provide designers some ability to trade circuit size for improved security.

Even though many other uses are possible, we focused on applications 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…

Fault injection attacks [22] are also ineffective against most of these applications, because the output and/or the input of the employed bit-mixers are not accessible to an adversary.

In the paper an overview of bit-mixers is provided, and a dozen of their information security (cryptographic) applications are informally discussed, ranging from key generators to whitening raw entropy, from improved ciphers to fast hash functions.

We invite further research on constructing bit-mixers and analyzing their use in security applications.

## II. Bit-Mixers

A Bit-Mixer is a function of fixed size input and output, computing *statistically* uncorrelated output from correlated input values. Its behavior is altered by parameters, called keys. More precisely, the 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 yields 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

Several families of bit-mixer constructions have been discussed in V, with power efficient, fast (single clock cycle) implementations for electronic hardware. They produce very little side channel leakage, as introduced in [2], [3] and [4].

Software implementations of bit-mixers from V are also orders of magnitude faster than cryptographic functions of similar parameters. Software bit-mixers may be based on different (e.g. RAX) constructions: [5] and [6].

## III. Applications of Bit-Mixers

Below we *informally* present a dozen information security applications where bit-mixers are beneficial. If not only the keys, but also the output and maybe even the input of the bit-mixers remain hidden from an observer, there are no special security requirements, or they are less stringent than at ciphers or other cryptographic functions, which are designed for high security even at visible input and output.

Hardware bit-mixers are very fast, and produce statistically uncorrelated output from correlated input, with low side channel leakage. By these properties bit-mixers can improve the speed and/or the security of cryptographic functions, when they perform certain internal computations.

### *A. Protected Memory in Secure Computers*

In this application bit-mixers are used as key generators for secure ciphers, e.g. AES [7]. A memory transaction unit between the RAM and the CPU transparently encrypts or decrypts data, and generates or verifies data authentication tags (stored together with the encrypted data). See in [8] and [9].

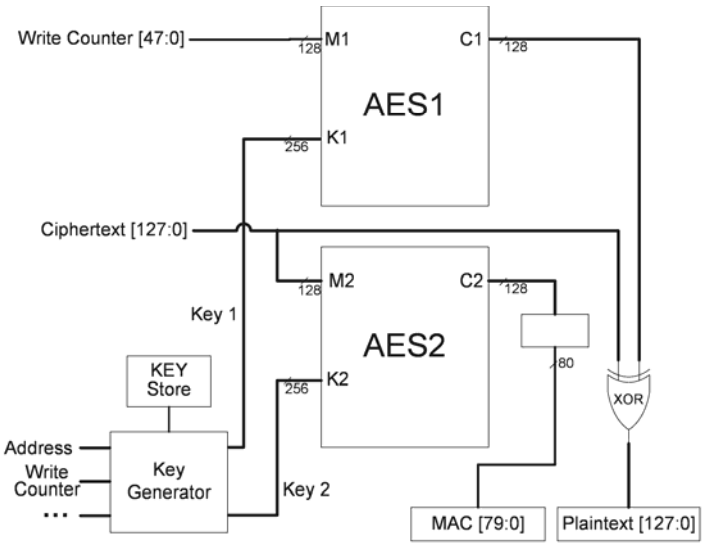


*Figure 2. Decryption in Protected Memory*

To prevent copying encrypted data to other memory locations, the encryption and authentication has to depend on the memory address.

To prevent memory rollback attacks, a global write counter is maintained, incremented at every memory write operation, while protected from illicit changes. The current write counter value also influences the generated keys. It has to be stored together with the encrypted data (like an IV) to enable decryption.

Note that individual memory blocks of encrypted and authenticated data could use their own write counters, but a single global write counter also offers the ability of testing the valid range of the read back values, for detecting tampering. See details in [20].

A fast implementation of a key generator for the memory protection feeds the memory address (e.g. 32 bits) and the write counter (e.g. 48 bits) into bit-mixers to *generate* 256 bit long encryption- and authentication-keys for a secure cipher, e.g. AES, using a large pool of secret key material. This way, each encryption and authentication uses a different, secret, statistically uncorrelated key – thwarting most side channel attacks, too.

During the startup of computing sessions the key material can be generated on-chip from physical random numbers (there is no need for sharing keys).

Figure 2 shows the block diagram of decryption in Protected Memory.

### *B. Key Rolling in Encrypted Communications*

In this application, as in the previous one, bit-mixers are used as key generators for secure ciphers, e.g. AES, but the setup and usage is different.

When many messages are encrypted and/or authenticated with constant keys, and an adversary can observe side channel leakage of the equipment of the sender or the recipient, then side channel attacks become feasible. They are mounted, e.g. by correlating data and the corresponding power traces, measured on the power lines of a security device, or analyzing EM emanations. To thwart these attacks, the encryption keys are changed after every-, or after every few messages, in a manner that both the sender and receiver can deduct the same key, but an adversary cannot (computed from secret keys).

Bit-mixers are ideal for this task: the input is an internal message index (counter), or a transmitted, non-repeating IV, and possibly the previous key and other, auxiliary information. Using shared secret key material, statistically uncorrelated secret keys are generated by both the sender and the recipient. All the input, the output and the keys of the bit-mixer remain hidden, thus only statistically good mixing is required.

### *C. Message Authentication Code with Data Dependent Keys*

An encrypted message is encrypted again with a secure cipher (or hashed, together with a secret key), and possibly truncated to form a MAC: a Message Authentication Code. It accompanies transmitted or stored messages [10].

There is no need for reversing these data authentication operations. When the authenticity of the message (ciphertext) is verified, the recipient repeats the computations of the MAC on the received message, and compares the result to the received MAC. They have to be the same to accept the message as authentic.

For a side channel attack, an adversary can systematically modify messages and feed them to the originally intended recipient, while observing side channel leakages of the corresponding computations of MAC values. Data bits and power traces could be correlated, when the MAC key remains constant.

To prevent these types of side channel attacks on the MAC computation, data-dependent MAC keys can be generated. Statistically uncorrelated keys make these side channel attacks infeasible, assuming that the key generation for the MAC has no significant side channel leakage.

Bit-mixers work for this key generation task, too: input is the data (ciphertext message) and possibly other, auxiliary information; output is the MAC key (e.g. for AES). Using shared secret key material, *statistically uncorrelated secret* MAC *keys* are generated, *dependent on the data*.

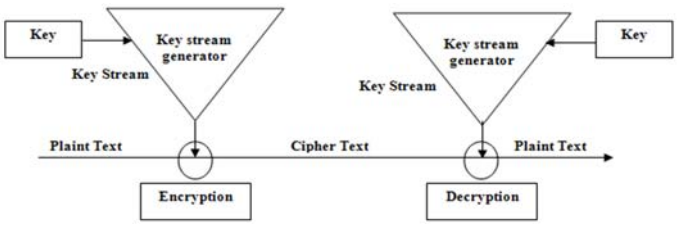


*Figure 1. Stream cipher*

### D. Parallel Stream Ciphers

Bit-mixers can be implemented by reduced rounds ciphers. It is straightforward to cascade such invertible bit-mixers (maybe different ones) to obtain true block ciphers. However, this way we lose any computational speed advantage. An alternative is *blending the output of a large number* of different *bit-mixers* (e.g. by bitwise XOR-trees with possibly some S-Box layers).

The input of these joint bit-mixers is formed by some previous outputs, maybe including a counter, as standard stream cipher modes specify [11].

The key material can be mostly hardcoded, consisting of arbitrary data without simple nonrandom patterns, and the cipher key is the first subkey, but there are many other ways to incorporate secrets in the key material and/or in the input.

The resulting joint bit-mixer functions are not invertible, but suitable for stream generation for stream ciphers. They use the same (pseudorandom) stream for encryption and decryption, as depicted on Figure 1. High speed is one of the advantages of these constructions. Another advantage is that the very short switching transients of the gates are hard to capture, and the parallel structure of hardware implementations mask side channel leakages very well.

The cryptographic security of the resulting stream cipher rests on the large number of parallel operating different bit-mixers, with each individual *output* is now *masked by* the output of *other bit-mixers*, and so each one remains hidden from an adversary.

### E. Scrambled-Counter Encryption Mode

The values of a repetitively incremented counter can be encrypted by a secure cipher (e.g. AES), using a secret key. This results in an arbitrary long bit stream, to be XOR-ed to messages, defining a specific stream cipher [12].

The encryption of counters values is an ideal target of side channel attacks, because the key is fixed, and most of the time a large part of the counter is constant (initialized by a practically never repeating IV). The other (counting) bits are known to an adversary.

A simple and fast remedy is using bit-mixers to *scramble the counter* values before they get encrypted, potentially mixing in previous outputs, too.

The key material can be mostly hardcoded, consisting of arbitrary data without simple nonrandom patterns, and the cipher key is the first subkey, but there are many other ways to incorporate secrets in the key material and/or in the input.

### F. Scrambled Tweak Encryption Mode

The XTS encryption mode [13] of AES on Figure 3 is an example of tweaked encryption: an encryption mode with modifications of the input and output values of a secure cipher by a secret key ($Key_2$), and by the position of the data block within a long message. The security is enhanced by reducing information leakage about repeated plaintext blocks, because the same data at different locations gets encrypted differently. Furthermore, data blocks cannot be moved around, without

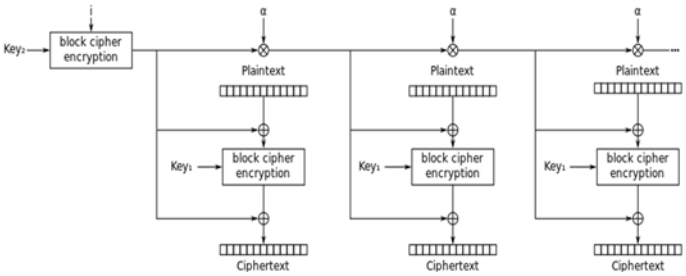


*Figure 3. XTS Encryption Mode*

completely randomizing the decrypted plaintext of the moved blocks.

The first tweak value is the encrypted location information of the data (or an IV), which is then repeatedly transformed by an LFSR (Galois multiplication with the polynomial $\alpha(x) = x$). Accordingly, half of the time, on average, a tweak value is a 1 bit rotated variant of the previous tweak, and in the other times a handful of the rotated bits are also flipped. Such simple correlations between consecutive tweak values make certain side channel attacks feasible.

*Replacing the LFSR operations* (×$\boldsymbol{\alpha}$) *with bit-mixers* reduces this vulnerability, without slowing down sequential implementations. The input of each bit-mixer is the previous tweak value, the output is the current tweak value, and the key material can be arbitrarily hardcoded, but without simple nonrandom patterns.

Using the data location as the input of the bit-mixer instead of the previous tweak, is equally secure, and allows parallel implementations: for the current tweak value only the location of the given cipher block within the message is needed, besides the secret key material.

### G. Pseudorandom Number Generator

Repeated calls of a bit-mixer with a counter or with the previous output as input yield statistically uncorrelated sequence of numbers. They are suitable as random numbers in technical computing, simulations.

Invertible bit-mixers mix a simple *counter* well, with guaranteed very long cycle, as opposed to re-mixing *previous output* (with unknown cycle lengths). A counter could replace a *part* of the output of an invertible bit-mixer, to form the input of the next mixing step. This gives very good mixing properties and guaranteed long cycles. See also in [5], [6].

Bit-mixer based compression functions may also provide security, when the pseudorandom number generator has a large internal state. This state is updated at every request for random numbers, and a much smaller output is computed from the internal state by a compression bit-mixer.

The key material can be hard coded, or the internal state of the random number generator may be used also as key material.

Many simple bit-mixers can operate in parallel, with their output blended together, e.g. by a bitwise XOR operations. Any single bit-mixer output is masked by the other bit-mixer outputs, so it remains hidden from an observer. This way an

adversary cannot find out even small parts of the internal state of the pseudorandom number generator from observed small joint output values, therefore this pseudorandom number generator is sufficiently secure.

One can also implement the NIST SP800-90A DRBG constructions [14] with two simple compression bit-mixers, for low security applications. Employing many such bit-mixers in parallel with their output combined (e.g. with bitwise XOR-trees with possibly some S-Box layers) yields higher security DRBG constructions.

### *H. Entropy Distillation (Whitening)*

True random numbers are needed for seeding and re-seeding pseudorandom number generators, also known as Deterministic Random Bit Generators (DRBG). Physical entropy sources are rarely perfect as they often yield biased and correlated bits. To reduce these non-randomness, one needs to mix the bits of the entropy source together, and then reduce the output size until the resulting bits are unbiased and statistically uncorrelated, and so they reach entropy close to 1 bit. Another alternative procedure is used in [19].

A compression bit-mixer with hardcoded key material and of the desired compression ratio can perform this whitening. It consumes very little computation time, when implemented in electronics (often in one clock cycle of an embedded system). See also in [14] and [15].

### *I. Round Key Generator for Ciphers*

Many ciphers have been broken due to their correlated round keys. Other ciphers are slow, spending too much computation to generate uncorrelated round keys. Hardware based bit-mixers offer remedies for both problems.

Constructs, similar to the bit-mixer based Pseudorandom Number Generator can be used to iteratively derive *round keys from the secret cipher key*. These improve the speed and/or the security of an iterative cipher C, which have too slow or too simple key schedule, resulting in a new cipher D, as follows.

1. Pick a bit-mixer B with the output size the same as the size of the round keys of cipher C, but the input size of B may be larger
2. Pick random key material M for the bit-mixer B. M can be chosen arbitrarily, avoiding simple non-random patterns
3. Distribute M among all communicating parties, intending to use the new cipher D
4. Replace the round-key generator of cipher C, with the bit-mixer B.

The employed bit-mixer can be used in different modes:

- The round number and/or the previous round key can constitute the input of B (with the original cipher key as the first input).
- The cipher key can be used as the first subkey of the bit-mixer B, and the round number is the input.
- A combination of the above constructions, and other similar ones could work, too.

Note that we have large freedom to choose the key material M for the bit-mixer B. With the choice of the key material one can *personalize ciphers*, that is, create as many different ciphers, as desired.

M can also be viewed as an additional, very long key for the new cipher D. In this case M is not hardcoded, but stored in registers.

### *J. Iterative Hash Functions*

Compression bit-mixers with hard coded key material can be directly used in Merkle-Damgård constructions, described in [16], as shown on Figure 5. This simple setup is suitable for high performance, low security applications (e.g. integrity checks of files, faster and still more secure than CRC).

For higher security, one can again use large numbers of bit-mixers working in parallel. Their outputs are combined (e.g. bitwise XOR-ed with possibly some S-Box layers). As we saw earlier, this parallel bit-mixer construction produces joint compression functions, with the output of any single bit-mixer hidden, masked by the output of the other bit-mixers, therefore the construction is sufficiently secure.

Bit-mixers also work in sponge constructions [17], which make use of compression functions, too. Simple bit-mixers are suitable for low cost, low security applications, while joint bit-mixers can replace the secure compression functions in cryptographic hash functions of the type SHA-3, see in [21].

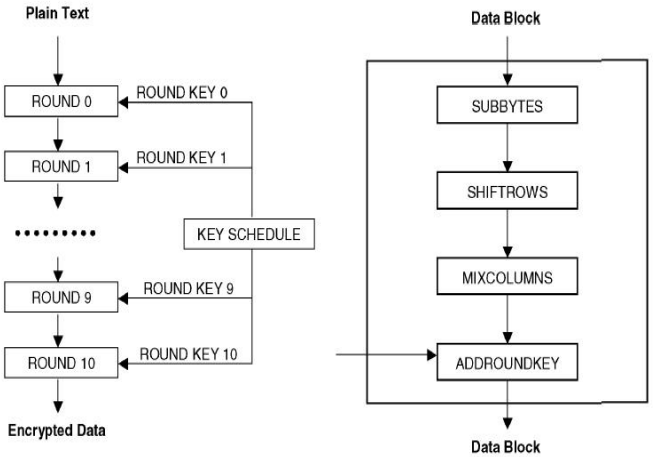


*Figure 4. Iterative cipher with key schedule*

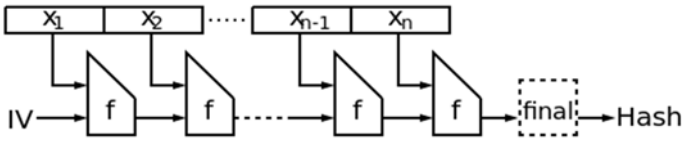


*Figure 5. Merkle–Damgård hash function construction*

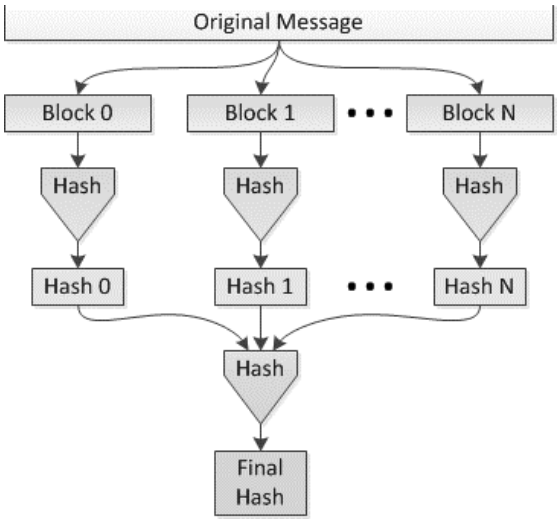


*Figure 6. Parallel hash function*

### K. Parallel Hash Functions

In certain communication systems (e.g. Internet protocols) the messages, to be hashed e.g. for integrity checks, may not become available sequentially. Message blocks can arrive at the recipient out of order.

There are also security platforms, which provide multiple compression functions, which can operate concurrently on different blocks of a message.

In these cases, the system could benefit from hash constructions, which can process blocks of the message out of order, or concurrently. Hardware bit-mixers allow us to construct such systems very efficiently, with little side channel leakage.

Bit-mixers are easily diversified. The same hard-coded keys can be used for bit-mixers of different architectures, or different keys can be used for similarly built bit-mixers. The simplest diversification of (extended input) bit-mixers can be done with using the same bit-mixers, but including a block number into their input.

Every block of the message to be hashed is processed by a diverse bit-mixer. Their outputs are combined, e.g. by a bitwise XOR-tree, or by an XOR-tree in where one or more layers are replaced with small (fast) compression S-Boxes, having fewer output bits, than input bits.

The corresponding construction is, faster and more flexible than the traditional parallel cipher construction depicted on Figure 6, (see also in [18]) where the combining simple hash function makes sure, that message blocks cannot be permuted without altering the overall massage digest. In our construction the diversity of the bit-mixers ensures that changing the order of the message blocks results in a different message digest (hash value).

For higher security, here again one can replace all simple bit-mixers with clusters of many bit-mixers working in parallel. Their outputs are combined (e.g. with bitwise XOR-trees with possibly some S-Box layers) to produce secure compression functions. As noted before, the output of any simple bit-mixer is now hidden, masked by the output of the other bit-mixers, making the construction secure.

### L. Message Authentication Code (MAC)

The bit-mixers in the iterative and parallel hash functions constructions, as discussed above, depend on sets of secret key material. If the key material is not hard coded, but shared between communicating parties, the hash functions described above become traditional MACs. See also in [10].

The key material can be generated from one or more smaller keys, e.g. by another bit-mixer or cryptographic expansion functions (e.g. the streams of cryptographic stream ciphers). *Nonlinear* bit-mixers (working in parallel or sequentially) are necessary for security (preventing an observer to build linear models, which could be solved with sufficient amount of data collected).

## IV. Optimizations Caveats

While the XOR-tree based bit-mixers can indeed be constructed at arbitrary input and output sizes, iterative bit-mixers (Substitution-Permutation networks, Feistel networks, Double-Mix Feistel networks or even Rotate-Add-XOR constructions, as described in V) are naturally created with equal input and output sizes $n$, called block length.

One may chose the block length $n$, larger than the required output size. There can be security reasons for it (not to use too small circuits, which an adversary can easily model) or practical reasons, including required long input sizes, or requirements resulting from sizes of intermediate values (which can be, e.g. multiples of the width of employed S-Boxes).

When $n$ is larger than the required output size, a few gates can be saved by not computing the output bits, which are not returned. This saving is normally insignificant, because only the last round can be simplified, for good mixing all bits are needed from previous rounds.

On the other hand, when the required input size $I$ is larger than $n$, one would be tempted to use one of the following tricks, or a similar one, to pack the extra $I-n$ input bits into a block of width $n$.

1. XOR the extra bits to the rest of the input, maybe to more than one other bit
2. XOR the extra input bits to the first (few) subkeys
3. Use small compressing S-Boxes for packing the extra bits to the rest of the input
4. Replace certain bits of the first subkeys with the extra input bits

Each of these simple bit packing strategies leads to the violation of at least the requirement 3 of bit-mixing: pairs, or groups of input bits are handled together, and so they can be modified, such that the output does not change. Therefore, bit-mixers of block size $n \geq I$, (not smaller than the required input size) must always be used, without any bit-packing tricks.

## V. Summary

Twelve areas of information security applications of the new concept of “bit-mixers” are presented, where the security and the processing speed are greatly improved by the bit-mixers. In these applications the output of individual bit-mixers are hidden from an adversary, making many attacks irrelevant. Nevertheless, the paper contains no rigorous security proofs, which will be included in future papers, where the parameters get fixed, including the block size, the key storage and the reuse strategy of the bits of the key material.

## References


[1] Laszlo Hars “Hardware Bit-Mixers.” Cryptology ePrint Archive, 2017.

[2] Kocher, Paul, Joshua Jaffe, and Benjamin Jun. "Differential power analysis." Advances in Cryptology—CRYPTO’99. Springer Berlin Heidelberg, 1999.

[3] Kocher, Paul (1996). "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems". Advances in Cryptology—CRYPTO’96. Lecture Notes in Computer Science 1109: 104–113. doi:10.1007/3-540-68697-5_9.

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

[5] 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. http://jes.eurasipjournals.com/content/2007/1/098417. doi:10.1155/2007/98417.

[6] L. Hars, G. Petruska: Pseudorandom Recursions II. EURASIP Journal on Embedded Systems 2012, 2012:1 doi:10.1186/1687-3963-2012-1. http://jes.eurasipjournals.com/content/2012/1/1

[7] J. Daemen and V. Rijmen. The Design of Rijndael. Springer Verlag, Berlin, 2002.

[8] United States Patent 8,839,001, “Infinite key memory transaction unit”.

[9] United States Patent 8,843,767, “Secure Memory Transaction Unit”.

[10] Bellare, Mihir, Ran Canetti, and Hugo Krawczyk. "Keying hash functions for message authentication." Advances in Cryptology—CRYPTO’96. Springer Berlin Heidelberg, 1996.

[11] Stream Cipher: https://en.wikipedia.org/wiki/Stream_cipher

[12] Lipmaa, Helger; Wagner, David; Rogaway, Phillip. "Comments to NIST concerning AES Modes of Operations: CTR-Mode Encryption". http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ctr/ctr-spec.pdf

[13] Morris Dworkin (January 2010). "Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices". NIST Special Publication 800-38E. National Institute of Standards and Technology.

[14] Elaine Barker, John Kelsey. DRAFT SP 800-90A. Recommendation for Random Number Generation Using Deterministic Random Bit Generators January 2012. http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf

[15] Elaine Barker, John Kelsey. DRAFT SP 800-90B (second draft), Recommendation for the Entropy Sources Used for Random Bit Generation, January 27, 2016 http://csrc.nist.gov/publications/drafts/800-90/draft-sp800-90b.pdf

[16] R.C. Merkle. Secrecy, authentication, and public key systems. http://www.merkle.com/papers/Thesis1979.pdf Stanford Ph.D. thesis 1979, pages 13-15.

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

[18] Goto, Eiichi, Tetsuo Ida, and Takao Gunji. "Parallel hashing algorithms." Information Processing Letters 6.1 (1977): 8-13.

[19] Peres, Yuval (March 1992), "Iterating Von Neumann's Procedure for Extracting Random Bits", Annals of Statistics 20 (1): 590–97, doi:10.1214/aos/1176348543

[20] United States Patent Application 20160063279, Hars; Laszlo: “Periodic memory refresh in a secure computing system”. March 3, 2016

[21] (NIST FIPS) – 202, Morris J. Dworkin “SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions” August 04, 2015 https://dx.doi.org/10.6028/NIST.FIPS.202

[22] 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.