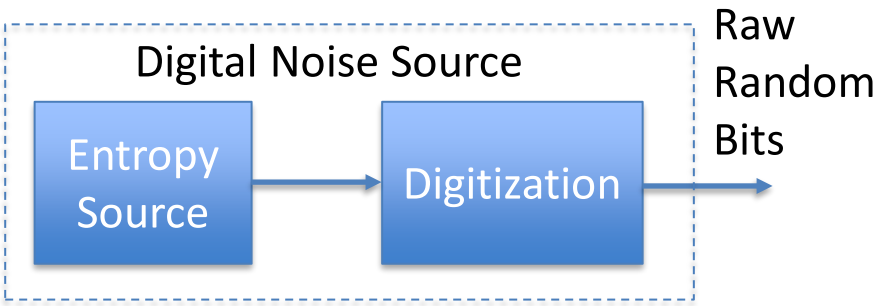
Postprocessing of Raw TRNG Bits

October 2021

Sujoy Sinha Roy

sujoy.sinharoy@iaik.tugraz.at



Raw random numbers produced in this way **are generally not IID**, i.e., independent and identically distributed.

- Bits are biased
- and contain correlation

Could we mitigate or remove statistical defects in raw random data?

Postprocessing (conditioning) of Raw Random Bits

‘Postprocessing’ is an application of a deterministic algorithm to removes or mitigates statistical defects from TRNG-produced raw random data (which contains defects).

- Increases randomness per bit by performing data compression.
- Some entropy is always lost due to data compression
- It doesn’t produce any ‘new’ randomness

Postprocessing (conditioning) of Raw Random Bits

‘Postprocessing’ is an application of a deterministic algorithm to removes or mitigates statistical defects from TRNG-produced raw random data (which contains defects).

- Increases randomness per bit by performing data compression.
- Some entropy is always lost due to data compression
- It doesn’t produce any ‘new’ randomness

There are two ways of postprocessing raw random bits:

1. Arithmetic postprocessing → do not rely on cryptographic primitives
2. Cryptographic postprocessing → rely on cryptographic primitives

Arithmetic postprocessing: Parity filter or XOR processing (1)

- Raw random bits are split into blocks of length n_f bits and
- Then the bits within each chunk are XORed

Example:

Raw bit sequence: 1 1 | 0 1 | 1 0 | 0 1 | 0 0 | 1 1 | 1 0 | 1 0 … with $n_f = 2$

XORed bit sequence: 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1

Arithmetic postprocessing: Parity filter or XOR processing (2)

- Raw random bits are split into blocks of length n_f bits and
- Then the bits within each chunk are XORed

Example:

Raw bit sequence: 1 1 0 1 1 0 0 1 0 0 1 1 1 0 1 0 ... with $n_f = 2$

XORed bit sequence: 0 1 1 1 0 0 1 1

Data compression factor is n_f.

If the raw data has a bias ϵ_{raw}

then the postprocessed data has a bias: $\epsilon = 2^{n_f - 1} \epsilon_{raw}^{n_f}$

Arithmetic postprocessing: Von Neuman Processing (1)

This method removes bias completely.

Steps:

1. Partition the input bit string into 2-bit blocks.
2. Discard all ’00’ and ‘11’ blocks.
3. If a block is ‘01’ then the output bit is 1; If a block is ‘10’ then the output bit is 0.

Example:

Raw bit sequence:	1 1	0 1	1 0	0 1	0 0	1 1	1 0	1 0	…
	×				×	×			
Output bit sequence:	-	1	0	1	-	-	0	0	

Arithmetic postprocessing: Von Neuman Processing (2)

This method removes bias completely.

Steps:

1. Partition the input bit string into 2-bit blocks.
2. Discard all ’00’ and ‘11’ blocks.
3. If a block is ‘01’ then the output bit is 1; If a block is ‘10’ then the output bit is 0.

Example:

Raw bit sequence: 1 1 0 1 1 0 0 1 0 0 1 1 1 0 1 0 …

Output bit sequence: - 1 0 1 - - 0 0

Output is produced at a variable rate.

If input has a throughput T_{in} then the average throughput of output is $T_{in} \cdot p_1 \cdot (1 - p_1)$.

Arithmetic postprocessing: Resilient Function [SMS07]

Definition [SMS07]: An (n, m, t)-resilient function is a function

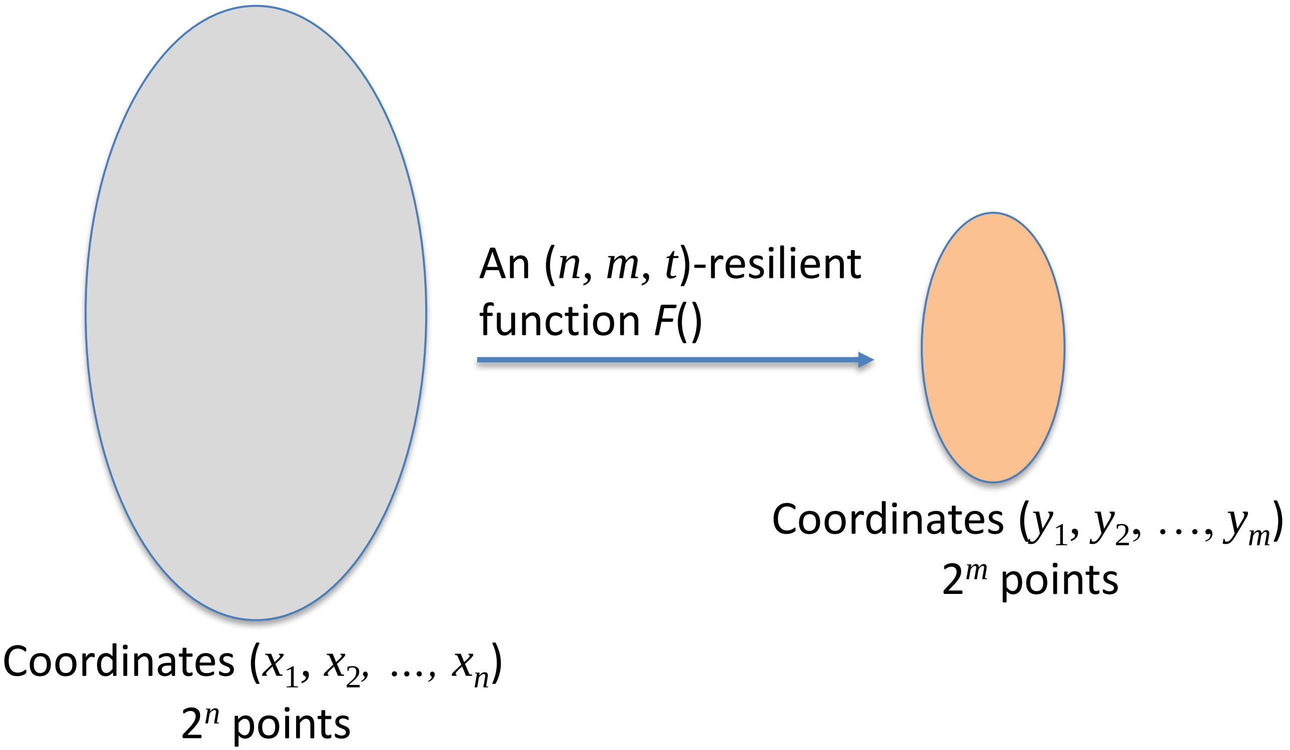
$$F(x_1, x_2, \ldots, x_n) = (y_1, y_2, \ldots, y_m)$$

from Z_2^n to Z_2^m enjoying the property that for any t coordinates $i_1, \ldots, i_t$, for any constants $a_1, \ldots, a_t$ from Z_2 and any element y of the codomain

$$Pr(\ F(x) = y \mid x_{i1} = a_1, \ldots, x_{it} = a_t\ ) = 1/2^m.$$

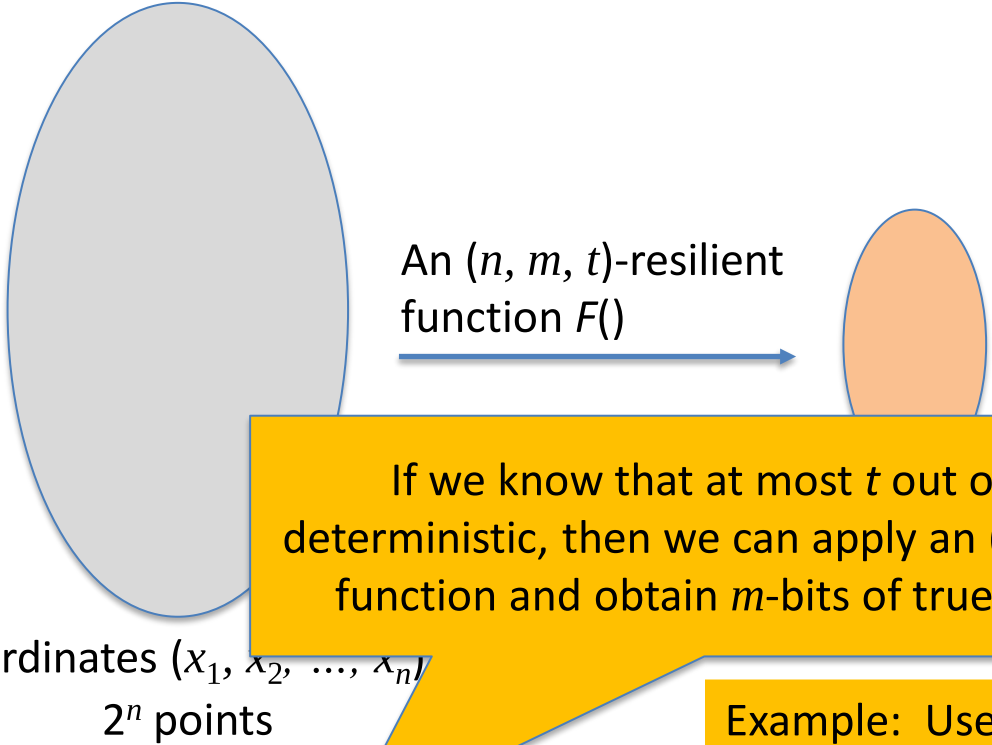
[SMS07] B. Sunar, W.J. Martin, and D.R. Stinson. "A Provably Secure True Random Number Generator with Built-In Tolerance to Active Attacks". IEEE Trans. on Comp., Vol. 56, No. 1, 2007.

Arithmetic postprocessing: Resilient Function [SMS07]



Knowledge of any ≤ t coordinates of input doesn't give any advantage in predicting output.

Arithmetic postprocessing: Resilient Function [SMS07]



If we know that at most t out of n bits are deterministic, then we can apply an (n, m, t)-resilient function and obtain m-bits of true randomness.

Example: Use an $(L, m, L/10)$-resilient function if 10% of the bits are deterministic.

Knowledge of any $\leq t$ coordinates of input doesn't give any advantage in predicting output.

Arithmetic postprocessing: Example of a Resilient Function

[SMS07] used a linear error correcting code $C = [n, m, d]$ to implement a $[n, m, d\text{-}1]$ resilient function.

$$f(x) = x \cdot \left(\quad G \quad \right)^T$$

This code can correct up to (d -1) “errors”

Arithmetic postprocessing: Example of a Resilient Function

[SMS07] used a linear error correcting code $C = [n, m, d]$ to implement a $[n, m, d\text{-}1]$ resilient function.

$$f(x) = x \cdot \begin{bmatrix} \\ G \\ \\ \end{bmatrix}^T$$

[SPV06] used a cyclic code for compact implementation on hardware platforms.

$$G = \begin{pmatrix} g_0 & 0 & \cdots & 0 \\ g_1 & g_0 & & 0 \\ \vdots & \vdots & \ddots & \\ g_{n-m-1} & g_{n-m-2} & \cdots & g_0 \\ 0 & g_{n-m-1} & \cdots & g_1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & & g_{n-m-1} \end{pmatrix}^T$$

[SPV06] D. Schellekens, B. Preneel, I. Verbauwhede. "FPGA Vendor Agnostic True Random Number Generator". IEEE FPL 2006.

Summary: Postprocessing (conditioning) of Raw Random Bits

'Postprocessing' is an application of a deterministic algorithm to removes or mitigates statistical defects from TRNG-produced raw random data (which contains defects).

- Increases randomness per bit by performing data compression.
- Some entropy is always lost due to data compression
- It doesn't produce any 'new' randomness

There are two ways of postprocessing raw random bits:

1. Arithmetic postprocessing → do not rely on cryptographic primitives
2. Cryptographic postprocessing → rely on cryptographic primitives

Cryptographic postprocessing

A cryptographic postprocessing uses a cryptographic primitive to process the raw random bits and then produce uniformly distributed random bits.

NIST recommended **keyed** algorithms for cryptographic postprocessing:

1. HMAC with any standardized hash function
2. CMAC with AES block cipher
3. CBC-MAC with AES block cipher

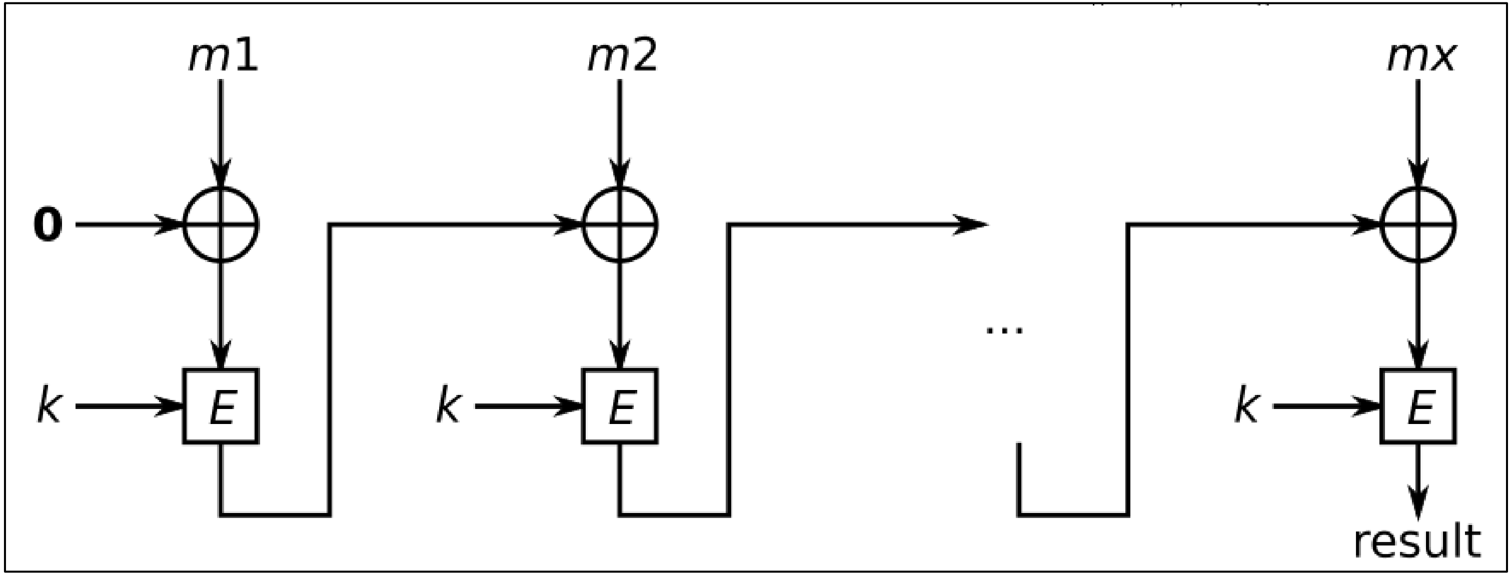
NIST recommended **un-keyed** algorithms for cryptographic postprocessing:

1. Any standardized hash function
2. Hash_df with any standardized hash function
3. Block_Cipher_df with AES block cipher

(Note: df stands for derivative function)

Cryptographic postprocessing: Example using CBC-MAC

Partition raw random bits into 128-bit blocks and use each block as a message-block.



E is AES-128.

The number of blocks ≥ 2.

Cryptographic postprocessing

Detailed technical information available on the NIST special publication SP 800-90A

NIST Special Publication 800-90A
Revision 1

Recommendation for Random Number Generation Using Deterministic Random Bit Generators

---

Elaine Barker
John Kelsey

References

[SMS07] B. Sunar, W.J. Martin, and D.R. Stinson. “A Provably Secure True Random Number Generator with Built-In Tolerance to Active Attacks”. IEEE Trans. on Comp., Vol. 56, No. 1, 2007.

[Yang18] B. Yang, "True Random Number Generators for FPGAs," PhD thesis, KU Leuven, 154 pages, 2018. https://www.esat.kuleuven.be/cosic/publications/thesis-307.pdf

[Rozic16] V. Rozic, "Circuit-Level Optimizations for Cryptography," PhD thesis, KU Leuven, 220 pages, 2016. https://www.esat.kuleuven.be/cosic/publications/thesis-286.pdf

[SPV06] D. Schellekens, B. Preneel, I. Verbauwhede. "FPGA Vendor Agnostic True Random Number Generator". IEEE FPL 2006. DOI: 10.1109/FPL.2006.311206