Creating an Efficient Random Number Generator Using Standard SRAM

A universal approach for adding strong cryptographic randomness to any IoT device

Geert-Jan Schrijen
Intrinsic ID
Eindhoven, The Netherlands
geert.jan.schrijen@intrinsic-id.com

Roel Maes
Intrinsic ID
Eindhoven, The Netherlands
roel.maes@intrinsic-id.com

Abstract— **With the rapid proliferation of Internet of Things (IoT) devices, it is vital to protect these devices and their communications. This requires implementing cryptographic systems suitable for low-cost devices, which include a Random Number Generator (RNG). Cryptographic protocols need random numbers, as certain inputs need to be unpredictable for attackers. The proper way to generate those is to first generate a truly random seed from a non-deterministic physical source. Then this seed is used as input for a deterministic algorithm that turns it into a large stream of random bits.**
A recent study from Bishop Fox [1] shows a critical vulnerability in RNGs used in billions of IoT devices, where the physical source fails to generate sufficient entropy. This compromises the RNG and puts the devices at risk of attack. This paper details a physical entropy source for creating a strong RNG solution that solves this problem, even on existing IoT devices.
When a chip is powered, its SRAM fills with a random pattern of 0s and 1s, which is highly unique for that chip and can be used as a device identifier (also known as a Physical Unclonable Function). However, between every powerup there also is a certain number of unstable bits in this unique pattern. A truly random seed for an RNG is created by harvesting this noise, without requiring changes to existing hardware. Given the wide availability of SRAM this solution can be added via software at low cost, making it perfectly suited for the IoT.

Keywords — Security; IoT; Cryptography; Random Number Generation; Embedded Software

I. Introduction

It is estimated that by 2025, there will be more than 27 billion devices connected to the Internet [2]. By interconnecting billions of devices on the internet of things (IoT), the world has become exposed to a plethora of security-related threats that never existed before. While companies struggle to recover from the damage caused by today's cyber-attacks, attackers are fabricating new low-cost attacks using increasingly cheaper tools to attack IoT devices. The need for more security is clear.
A recent publication [3] from the Global Semiconductor Alliance (GSA) outlines the concept of a security subsystem, integrated into a larger microcontroller or system-on-chip (SoC) controller, which in turn is at the heart of an IoT device. Applications use this security subsystem for services such as encryption of sensitive data, device authentication and setting up secure connections to other devices.

To provide support for such applications, security subsystems typically implement cryptographic functions for encryption, message authentication and digital signatures. The proper use of these functions requires a unique set of cryptographic keys that are accessible only within the security subsystem and hence stored securely from outside observers. Additionally, unpredictable random values are needed for various cryptographic protocols that the security subsystem is handling.

A. *Secure Key Storage*

Cryptographic keys are essential to the use of cryptography. It was Auguste Kerckhoff who stated already in the 19th century, that the security of a cipher should rely only on the secrecy of the key and not on the secrecy of the cipher[4]. This principle has become known as Kerckhoff's Principle and is the basis of all modern-day cryptography.
The fact that security relies totally on the security of the used cryptographic keys has an important implication in practice. It is of vital importance to guarantee that the cryptographic keys used by a security subsystem are securely stored and only accessible within the security subsystem. They need to be guarded against readout, alteration and copying by attackers. Traditional methods for secure key storage rely on storing a root cryptographic key in non-volatile memory such as embedded flash memory or one-time programmable (OTP) memory. However, these methods have serious disadvantages in terms of security, cost, reliability, and flexibility. A novel approach

avoiding these downsides is the use of an SRAM Physical Unclonable Function (PUF), as is described in several publications [5][7].

B. Random Number Generation

Besides relying on a well-protected secret key, many cryptographic protocols also depend on the availability of unpredictable random numbers. They are for example needed in encryption schemes as initialization vector, for key establishment protocols, and for the generation of secret keys, PINs and passwords. A bad random number generator would lead to predictability in the generated keys, which would directly reduce the security level of a cryptographic mechanism using such keys and hence give an undesirable advantage to an attacker.

1) Problems with random numbers in practice

Over the past years there have been several examples where bad quality random numbers have resulted in serious security problems.

In 2010, a group of hackers called “fail0ver” discovered a serious flaw in the use of random numbers for digitally signing the Sony PlayStation 3 (PS3) software [8][9]. In the Elliptic-Curve Digital Signature Algorithm (ECDSA) it is required that for every signature that you generate, a new random nonce value (the “k” parameter) is used. Failing to do so can be catastrophic, as the private signature key can be directly derived from only 2 signed messages that use the same nonce. Unfortunately, that is exactly the flaw in Sony’s implementation; they signed their software with a constant nonce. Hence, with only two signed firmware images, the hackers were able to retrieve Sony’s signing key and they could correctly sign their own modified software to take over the device.

A related problem of insufficient randomness in the application of ECDSA signatures was found in 2013 in Android implementations of bit-coin wallets[11]. Due to issues with the underlying random generator in the java-based random generator of Android, (pseudo) random sequences were occasionally repeating. This led to ECDSA signatures on bitcoin transactions with the same nonce values, leading to compromised signature keys and theft of money.

The security of the RSA algorithm relies on the difficulty of factoring large integers. RSA public keys are constructed by multiplying two large prime numbers. These prime factors form the private key of the crypto system and need to be of large enough size and randomly generated in order to be unpredictable to an attacker. In 2012 researchers discovered issues with the RSA keys of thousands of Internet connected devices [10]. It turned out that these devices have public keys that share the same prime factor as part of their public key as other devices. Whereas it is computationally infeasible to factor a large composite number consisting of two random and sufficiently large prime numbers, feasibility dramatically increases when two RSA keys share a common prime factor because more efficient algorithms exist to compute the greatest common divisor of a product. Hence, the feasibility of recovering the prime factors in RSA keys drastically increases when RSA keys share common prime factors, and this caused an immediate problem for the security of the discovered devices relying on such weak keys. The underlying problem of these weak RSA keys is the fact that they had been generated using weak random generators on those devices.

More recently, a study of security company Bishop Fox showed issues with the use of random generators in IoT devices[1], putting billions of devices at risk. The researchers found that in many cases the main microcontroller in an IoT device does have a built-in true random number generator, but that it often has shortcomings that the developer is not aware of. For example, hardware random number generators cannot always produce large streams of random numbers at high speed and will indicate when they fail to do so. However, such return codes need to be checked and software must take care of proper handling. Unfortunately, the researchers found several cases of widely re-used code in embedded operating systems where return codes are not checked at all. Furthermore, some hardware random generators have certain prescriptions of use which are often overlooked or turn out to have imperfections in the entropy that they produce.

2) The right approach

The proper way of establishing a cryptographically secure random number generator on embedded devices is to start with a true-random entropy source in the hardware of a device that seeds a deterministic random bit generator (DRBG). The entropy source must have known randomness properties such that a guaranteed full entropy seed can be extracted to initialize the DRBG. Furthermore, built-in health checks should run upon initialization to monitor entropy of the source and halt upon detection of deviations from the expected quality. The DRBG produces large streams of random numbers to calling applications and will trigger a reseed from the entropy source after a pre-described number of output bits. Such an approach is documented in the NIST SP 800-90 standards [12][13][14]. Following these standards is a requirement for devices that claim FIPS compliance [15].

C. Outline

In the remainder of this paper, we describe how SRAM based Physical Unclonable Functions (PUFs) can be used to build a strong cryptographic random number generator on embedded devices. We first describe the properties of SRAM PUFs and how they can be used as a source of true randomness. Then we explain how to use an SRAM PUF source to build a FIPS compliant random generator according to the NIST SP 800-90 standards. Finally, we show an example validation of the SRAM PUF based entropy source using the NIST statistical test suite on measurements obtained from an actual microcontroller device.

II. The SRAM PUF

Due to deep submicron manufacturing process variations, every transistor in an Integrated Circuit (IC) has slightly different physical properties. These lead to small but measurable differences in terms of electronic properties such as transistor threshold voltage and gain factor. Since these process variations are not controllable during manufacturing, these physical device properties cannot be copied or cloned. Threshold voltages are susceptible to environmental conditions such as temperature so their values cannot be used directly as unique secret keys or identifiers.

The PUF behavior of an SRAM cell, on the other hand, depends on the difference of the threshold voltages of its transistors. Small differences will be amplified and push the SRAM cell into one of two stable states. Its PUF behavior is therefore much more stable than the underlying threshold voltages, making it the most straightforward and stable way to use the threshold voltages to build an identifier.

A. *SRAM PUF Behavior*

An SRAM memory consists of an array of SRAM cells. Each SRAM cell consists of two cross-coupled inverters that each are built up by a p- and n-MOS transistor, see Figure 1. When power is applied to an SRAM cell, its logical power-up state is mainly determined by the relation between the threshold voltages of the p-MOS transistors in the invertors. The transistor with the smallest threshold voltage will start conducting first and determines the outcome, a logical '0' or '1'.

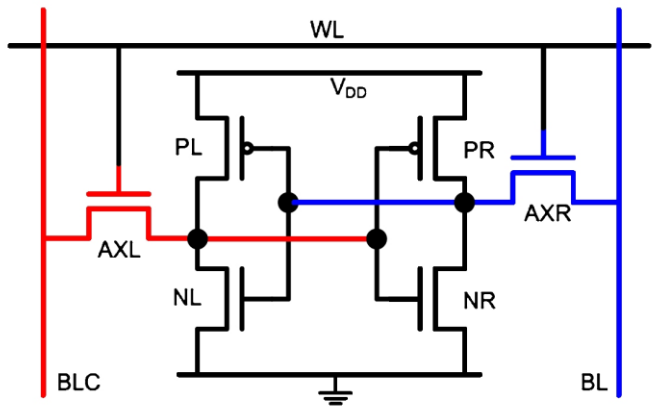

Figure 1: Schematic of 6-transitor SRAM cell. The left inverter consists of PMOS transistor PL and NMOS transistor NL and is cross-coupled with right inverter consisting of transistors PR and NR. Transistors AXL and AXR are access transistors for read and write operations. The SRAM cell is accessed through word line WL and bit line BL or complementary bit line BLC.

It turns out that most SRAM cells have their own preferred state every time the SRAM is powered resulting from the random differences in the transistor threshold voltages. This preference is independent from the preference of the neighboring cells and independent of the location of the cell on the chip or on the wafer.

Hence, an SRAM region yields a unique and random pattern of 0's and 1's that is stable for most of the bit cells.

A small fraction of the bit cells happens to have threshold voltages in the cross-coupled inverters that are closely matched. These cells will sometimes power up as a logical '0' and sometimes as a logical '1'. Hence, these bit cells produce noisy results at every power-up.

The combined power-up pattern of an SRAM memory hence consists of a majority of stable cells with a unique pattern and a small fraction of noisy bit cells. An SRAM PUF response can therefore be regarded as a "noisy fingerprint" of a device.

The amount of noise between consecutive SRAM PUF measurements at room temperature is typically in the order of 5%, see for example Figure 2 and Figure 3. When comparing measurements from different temperatures to a reference measurement at room temperature, the relative noise is typically higher and can go up to 15% at extreme temperatures. In the measurement data depicted in Figure 2, the largest difference compared to room temperature is still less than 11% and occurs at +125°C. This difference needs to be error corrected by the Fuzzy Extractor when deriving cryptographic keys from the SRAM PUF. Various studies have investigated SRAM PUF in more detail, see for example [23][24][25].

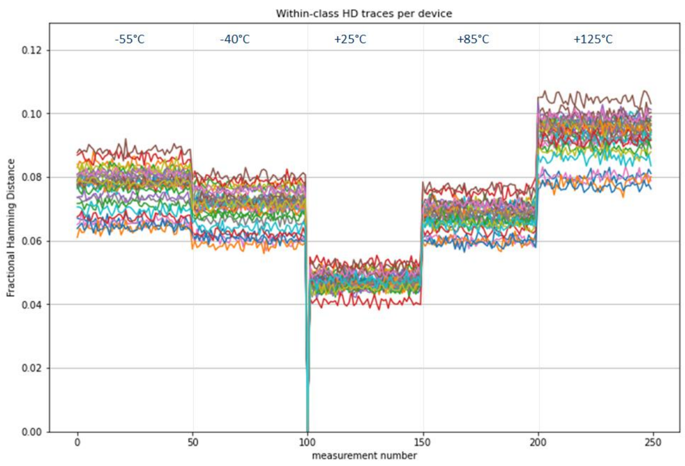

Figure 2: SRAM PUF measurements from 30 devices in UMC 65nm technology node, depicting the Within-Class Hamming Distance (WCHD) between measurements of the same device at different temperatures and a reference measurement at room temperature (first measurement at 25°C). At every temperature, 50 measurements were taken per device. WCHD levels at room temperature are between 4% and 6%. At more extreme temperatures the WCHD compared to room temperature increases up to 11%.

When using the SRAM PUF to generate randomness entropy, we are not interested in the Hamming Distance with respect to a reference at room temperature, but rather in the PUF noise when comparing measurements to a reference of the same device at the same temperature. This will give an indication of the smallest amount of noise that can be expected and hence provides the minimum amount of noise entropy. Figure 3 shows the SRAM PUF noise levels at different temperatures.

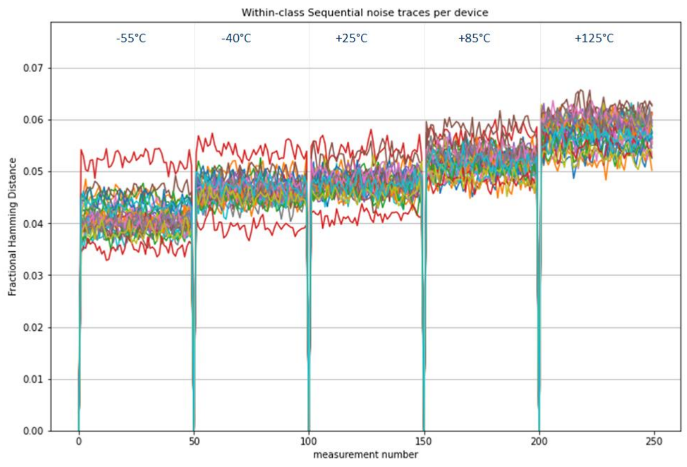

Figure 3: SRAM PUF noise measurements from 30 devices in UMC 65nm technology node, depicting the Within-Class Hamming Distance (WCHD) between measurements of a device and a reference measurement at the same temperature (first measurement at each temperature). At every temperature, 50 measurements were taken per device. SRAM PUF noise levels are relatively stable over temperature. Lowest noise levels occur at the lowest temperature.

It is observed that the noise levels are relatively stable over temperature. At -55°C SRAM PUF we see the lowest noise levels of the temperature range, which are between 3.5% and 5.5%. These are only slightly lower than the noise levels observed at room temperature (+25°C), which are between 4% and 6%.

B. Key Generation and Storage Based on SRAM PUF

Keys that are derived from the SRAM PUF are not stored 'on the chip' but they are extracted 'from the chip', only when they are needed. In that way they are only present in the chip during a very short time window. When the SRAM is not powered there is no key present on the chip making the SRAM PUF keys very difficult to attack.

How does it work? To derive a cryptographic key from a PUF, a so-called Fuzzy Extractor [16][17][18] is needed to turn the slightly noisy PUF response of the chip into a reliable root key. The two main algorithms inside a Fuzzy Extractor are:

1. Error Correction: to correct the noise on a measured PUF response by applying error correcting codes. So-called helper data is stored to provide additional information for the error correction. It is constructed in such a way that it does not leak any information on the reconstructed root key. The error correction guarantees that under any circumstances that influence the noise (such as extreme temperatures), the device's root key can be reconstructed reliably.
2. Privacy Amplification: to guarantee full entropy of the output root key, despite the information present in the helper data. After error correction the data is compressed into the actual root key, e.g., of 256 bits.

Whenever the root key is needed by the system, the Fuzzy Extractor runs its reconstruction operation by reading the SRAM power-up values and the helper data generated at an initial one-time enrollment step. There is no need to store this root key in any form of non-volatile memory. This means that when the device is powered off, no secret key can be found in any memory; in effect, the root key is "invisible" to hackers. A whole tree of cryptographic keys (starting from the PUF root key) can be (re-)created without storing them in a memory, removing the need for a device to have any physical form of secure storage. More details about the basic functionality of SRAM PUF can be found in "SRAM PUF: The Secure Silicon Fingerprint" [6].

C. Randomness Generation Based on SRAM PUF

Besides extracting keys from the SRAM PUF responses, the SRAM PUF responses can also be used as a source of non-repeating true randomness. The randomness comes from the small percentage of bit cells in an SRAM memory, whose transistor threshold voltages happen to match quite closely. These bit-cells do not have a strong preferred power-up state but tend to power-up randomly: sometimes as a logic '0' and sometimes as a logical '1'. The stability of PUF response bits from an SRAM memory can be investigated by analyzing their so-called one-probability.

One-probability is defined as the probability that an SRAM cell powers up in the logical "1" state. As can be seen from the probability density histogram in Figure 4, most SRAM PUF cells have a one-probability value either close to 0.0 or close to 1.0., which respectively indicates a stable 0-producing cell or a stable 1-producing cell. Only relatively few cells will have a one-probability not close to either 0.0 or 1.0, indicating an unstable cell, i.e., a cell for which the response value resulting from a power-up is not relatively certain upfront, and which hence has some level of unpredictability to it. A small (but non-negligible) minority of cells will have a one-probability close to 0.5, which indicates a fully unpredictable response behavior. The unstable cells are responsible for the so-called noisy behavior of the SRAM PUF, generating fresh entropy upon every power-up of the SRAM memory. This entropy is also called the noise entropy of the PUF.

Analyses of actual SRAM PUF measurements show that about 5% of the cells will have a noise min-entropy contribution of more than 0.5 bit/cell. As a consequence, for an SRAM PUF which contains a sufficiently large number of individual cells, there will always be a considerable number of cells which consistently generate noise entropy (e.g., in an SRAM array of 1000 cells, there will be on average about 50 cells that generate a significant level of noise min-entropy on every power-up). This noise entropy provides the basis for using the SRAM PUF as a noise source in a FIPS compliant random number generator.

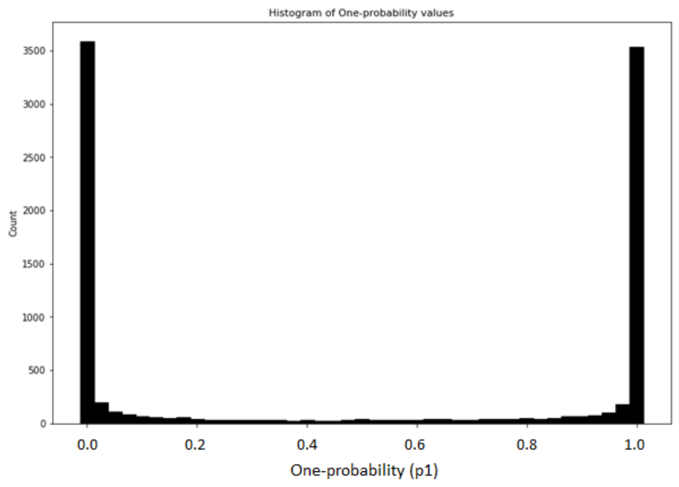

Figure 4: One-probability distribution of a 1KB SRAM array measured 1000 times at room temperature. Most of the bit cells power up as a very stable digital 0 or 1 value (resulting in the peaks in the above histogram). A minority of bit cells has a more random power-up behavior and contribute to the noise entropy.

Considering the typical one-probabilities as observed in SRAM PUF responses, we can say that the noise entropy contained in them is sparse and diluted:

- The total noise entropy produced by an SRAM PUF array will mostly be generated by a minority of its cells that contribute a relatively high entropy rate. These few entropy- contributing cells are distributed ***sparsely*** over random positions in the SRAM PUF array.
- The total amount of noise entropy produced by an SRAM PUF array will be relatively low compared to the size of the array in terms of number of cells. The expected (averaged) entropy contribution per cell will hence be rather low, or in other words the noise entropy in an SRAM PUF response is ***diluted***.

Because of these properties, we propose to use an additional entropy concentration function on the SRAM PUF output such that there is a better fit with the NIST entropy source model, as is explained in the next section.

III. FIPS Compliant Randomness Generation

FIPS 140-3 compliant security modules need to have a random number generator that is compliant to the NIST SP800-90 specifications [12][13][14]. According to these specifications, an approved random number generator consists of a Deterministic Random Bit Generator (DRBG) that is requesting entropy from a randomness source such as a NIST approved Entropy Source, see the setup in Figure 5.

In the following subsection we will explain how an SRAM PUF can be used to create an Approved Entropy Source according to the NIST SP800-90B recommendations.

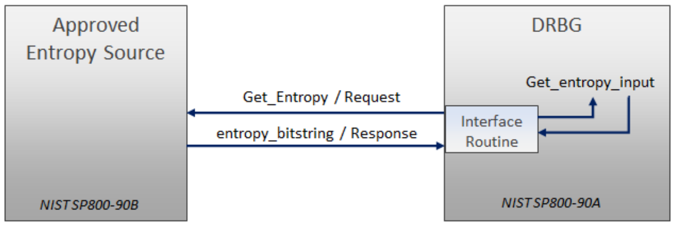

Figure 5: Schematic architecture of an approved Random Number Generator construction according to the NIST SP800-90C specification [14].

A. *Entropy Source*

Guidelines for NIST approved entropy sources are given in the NIST SP800-90B specification [13]. There an entropy source is considered to include the components as depicted in Figure 6. It comprises a Digital Noise Source, whose output is Raw data that is being tested with a Health Test function. Before the Raw data is output, it is optionally conditioned. The Digital Noise Source consists of an Analog Noise Source whose output is digitized using a Digitization function.

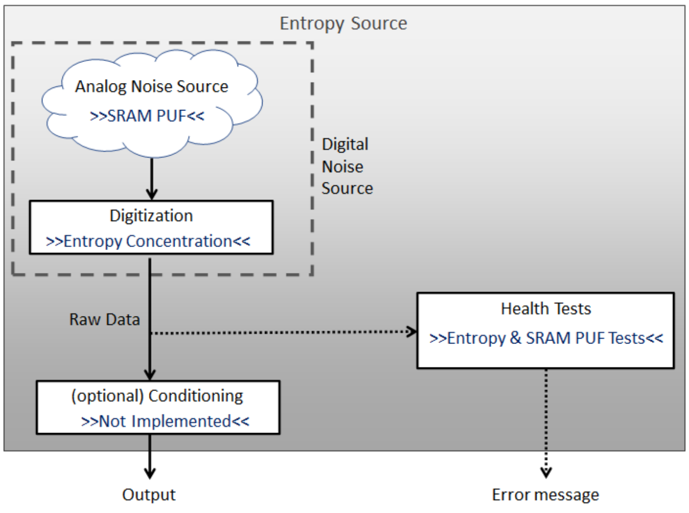

Figure 6: Entropy Source Model according to NIST SP800-90B specification [13]. Texts between >>marks<< indicate the proposed implementation of components for an SRAM PUF based entropy source

.

1) *Analog Noise Source*

To create an SRAM PUF based entropy source, we can use the power-up state of SRAM as the “Analog Noise Source”. Even though the direct evaluation of the SRAM PUF is already in a digital (binary) form, an additional “digitization” step is proposed to transform the noise source output into digital noise samples that have the required properties such as a guaranteed lower-bound on the min-entropy.

2) *Digitization*

The reason for this proposed digitization step is the fact that the noise entropy present in the binary PUF is sparse and diluted, as was discussed in the previous section. The NIST SP800-90B specification requires that each individual sample produced by the noise source should have a guaranteed lower-bound for its (min)-entropy. If the noise source produces (occasional) samples with very low entropy, it will fail some of the statistical validation checks defined by the specification, even if the total collection of produced samples as a whole contains sufficient entropy for the intended application. Using SRAM PUF bits directly as noise samples is hence suboptimal because many bit locations do not contain any noise entropy at all (due to the sparseness).

We propose to implement an Entropy Concentration function as "digitization" step, which ensures that the sparse and diluted noise entropy in the SRAM PUF bits is transformed into a smaller set of output bits in which the noise entropy is concentrated. This entropy concentration function should not obfuscate the statistical properties of the physical behavior on which the noise source relies as the origin of the noise entropy. This rules out the use of, e.g., cryptographic hash functions, and most other kinds of cryptographic operations, which would typically be used to extract values with a high entropy density from sources with a low entropy rate.

A simple binary matrix multiplication can do the required job. The matrix can be designed with the property that when used in the binary multiplication, it will mix a large number of input bits into a small number of output bits, while preserving to a large extent the entropy of the input. This guarantees that noise in the sparsely distributed noisy input bits gets concentrated into a smaller number of output bits. Moreover, such a matrix multiplication is a simple linear transform which largely retains the statistical properties of the noise source, allowing for meaningful testing and validation of the noise source samples. An example of a suitable matrix with the desired properties, is the parity-check matrix of a Reed-Muller code.

3) *Health Tests*

The output of the Digital Noise Source, the Raw Data, needs to be tested by means of Health Tests to detect deviations from intended behavior. The goal of these tests is to ensure that the noise source operates as expected (under potentially varying external conditions) and that critical failures of entropy generation are detected. Health tests need to be tailored to the specific noise source and are typically technology specific. They are expected to raise an alarm when there is a significant decrease in the entropy of its outputs, when noise source failures occur, or when underlying hardware fails.

The NIST specification [13] requires that both startup tests and continuous tests are included. Whereas startup tests run after powering up or rebooting to verify that the noise source components are operational, continuous tests need to run when the noise source is operating to detect failures while the noise source produces outputs. An SRAM PUF based random source does not continuously produce random outputs, but instead delivers entropy at power-up of the SRAM memory. So that is the moment at which the "continuous" tests as described by NIST need to be run.

Two statistical tests on the Raw Data that are mandatory according to NIST SP800-90B are:

- Repetition Count test: tests for too long sequences of repeating noise sample values,
- Adaptive Proportion test: tests for a too high occurrence of a noise sample value in a fixed-length window.

Both these tests as described in the specification can be applied directly to the Raw Data as produced by the SRAM PUF based noise source. Additional vendor-defined tests are recommended to test technology specific failure modes which are not sufficiently covered by the two mandatory tests.

In the case of an SRAM PUF based noise source we propose to add tests on the actual PUF response values before the digitization step, to increase the sensitivity of detecting problems. The Repetition Count and Adaptive Proportion tests can (with adapted parameters) be extended to be run on PUF response values directly.

Additionally, to detect reuse of SRAM PUF values without proper re-powering in between (a very specific SRAM PUF failure mode), we propose to add an SRAM state test. After using the SRAM PUF values for harvesting entropy, a pre-defined value is written in a pre-defined part of the SRAM. When entropy is requested, it is first checked whether the pre-defined part of SRAM contains the pre-defined value (indicating it has already been used) or not. This way re-use without proper repowering can be detected and avoided.

4) *Conditioning*

The conditioning function is optional according to the NIST specification. It can be used to reduce bias and/or to increase the entropy rate of the output bits. However, when used in combination with a DRBG mechanism which anyway cryptographically compresses the entropy input upon instantiation, a conditioning function in the entropy source is generally not needed.

5) *Interfaces*

At least the following conceptual interface functions should be implemented in an approved entropy source:

- GetEntropy: an interface over which the consuming DRBG can request entropy and in return obtain a bitstring containing at least the requested amount of entropy. After checking that the SRAM PUF values have not been used before (SRAM state test), the PUF response values are passed through the digitization function. After applying the health tests as described in the previous subsection, the Raw Data is provided via the output to the calling DRBG. To prevent accidental reuse or disclosure of SRAM PUF response values, the SRAM can be zeroized.

- GetNoise: an interface to obtain raw, digitized outputs from the noise source for use in validation testing or external health testing. This function follows the same procedure as the GetEntropy function, but without executing the health tests. The function should be made available in a special test mode and not be callable in a regular operational mode. Calling GetNoise prohibits any further calls to GetEntropy, until the SRAM (or the device) is properly repowered.
- HealthTest: an interface over which the internal health tests can be triggered. This function only outputs an okay or not-okay signal to the calling application, without actually outputting any random data. The GetEntropy function can still be called after calling this function as it does not destroy the noise entropy in the SRAM (i.e., no zeroization is applied).

B. *DRBG*

The Deterministic Random Bit Generator (DRBG) is the main mechanism that is called by the application to deliver random data. Approved cryptographic mechanisms are described in the NIST SP800-90A specification [12]. The implementation of a DRBG according to this specification does not depend on the specifics of the noise source used. Hence there are no specific mechanisms that need to be implemented related to our SRAM PUF based noise source. The functional model for a DRBG as described by NIST is depicted in Figure 7.

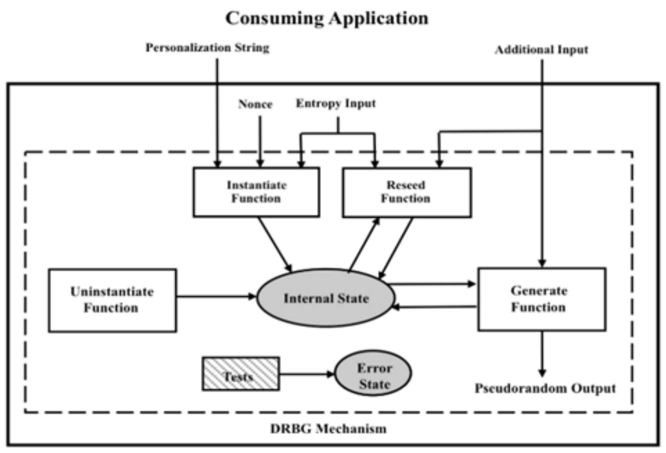

Figure 7: DRBG Functional Model, as taken from the NIST SP800-90A specification [12].

A consuming application requests random bits by calling the Generate Function. Application specific input data can be mixed in via an optional Additional Input if desired. The Generate Function uses a NIST approved pseudo-random function of the proper security strength for updating its state and generating output data. Correct implementation of the used pseudo-random function can be validated and certified via the NIST CAVP program [20].

Before being able to call the Generate Function, the initial state first needs to be created by calling the Instantiate Function. The Instantiate Function brings in true randomness from the Entropy Source and mixes that with an optional Personalization String input provided by the application.
The NIST specification requires the presence of an Uninstantiate Function to destroy the Internal State in case of certain security related events such as detected breaches. Furthermore, there needs to be a Test function to test the correct operation of the DRBG, which can be called at any time and is automatically triggered upon initial use of the DRBG.

The Reseed Function is optionally implemented to have the possibility to provide additional entropy that will securely update the internal state of the DRBG. This function could be used to restore secrecy of the internal state when full secrecy of the internal state cannot be guaranteed anymore. The NIST specifications prescribe that a seed has a limited lifetime. In particular, the maximum number of requests that can be served from a single seed is 2^{48} for DRBGs that use a SHA or AES based pseudo-random function. An internal counter is keeping track of the actual number of requests that have been handled and the DRBG shall indicate that a reseed is required when the maximum number of requests is passed.

1) *Special Considerations*

Two limitations when working with an SRAM PUF based entropy source in this respect are:

1. After the DRBG has been uninstantiated, a repower of the SRAM (or the device) is needed before a new instantiation can take place, to generate fresh noise entropy for the noise source.
2. A reseed can only be implemented from the SRAM PUF based noise source when the used SRAM memory has a dedicated SRAM power switch. In case the SRAM is powered along with the rest of the device, a reseed cannot be implemented from the same entropy.

In practice, the reseed limitation is not an issue as typically multiple random bytes are requested at once (per request) and 2^{48} requests can be served without reseeding. This huge number of requests is not even reached within 100 years of device operation when the DRBG would be called once every millisecond.

IV. Entropy Source Validation

The NIST SP800-90B specification [13] describes how entropy sources can be validated with statistical tests, to assure that the relevant requirements of the specification are met. This is a procedure that is typically executed by an accredited laboratory. The entropy source vendor needs to provide an entropy estimate, which is based on its own model and analysis of the noise source. In this section we provide an example on how this is done.

For the SRAM PUF noise source, we can use our probabilistic model of an SRAM PUF [21] to provide an initial entropy estimate. The model is tuned to an actual device implementation by measuring the SRAM PUF noise and bias (see section “The SRAM PUF”) from a platform on which we want to implement our random number generator.

In this example case, we use measurements from a Texas Instruments TM4C123GH6PM microcontroller. At a constant temperature we obtain 1000 PUF measurements of an 8 kilobyte SRAM region by repeating the following procedure 1000 times:

1. power off the board,
2. wait 1 second,
3. power on the board
4. read out SRAM contents and write it to a file on the controlling PC.

The PC is used to program the microcontroller and to retrieve measurement data over the UART/USB interface. The measured PUF noise and one-probability distribution (before digitization) are used as input to our PUF model to compute an entropy estimate.

For the digitization step we implement an entropy concentration function in the form of a matrix multiplication with the transposed parity check matrix of a Reed-Muller(5,8) error correcting code. With this matrix multiplication we transform every block of 256 input bits from the SRAM PUF into 37 output bits that serve as raw samples of the noise source. After applying this entropy concentration function, the one-probability distribution of the output bits looks as is depicted in Figure 8. The figure shows that after applying the transformation, most of the bits have a one-probability that is close to 0.5. Compare that to the original one-probability distribution that was presented in Figure 4.

Using our PUF model, we find a lower-bound entropy estimate of 0.37 bits per raw noise sample, where a sample equals a single output bit after applying the entropy concentration function. This value is used as the “submitter entropy” value for the statistical entropy validation in the next stage.

Next, we run the NIST-SP800-90B Entropy Assessment test suite [22]. This test suite is used to validate with statistical tests, which do not have knowledge of the specific noise source model, that there are no statistical indications of any entropy issues. For the statistical tests, two datasets are created based on the measured SRAM PUF data:

1. A sequential dataset, where we concatenate all bits of the measurements after applying the entropy concentration function,
2. A restart dataset, where we use exactly 1000 output bits (after applying the entropy concentration function) for each of the 1000 measurements.

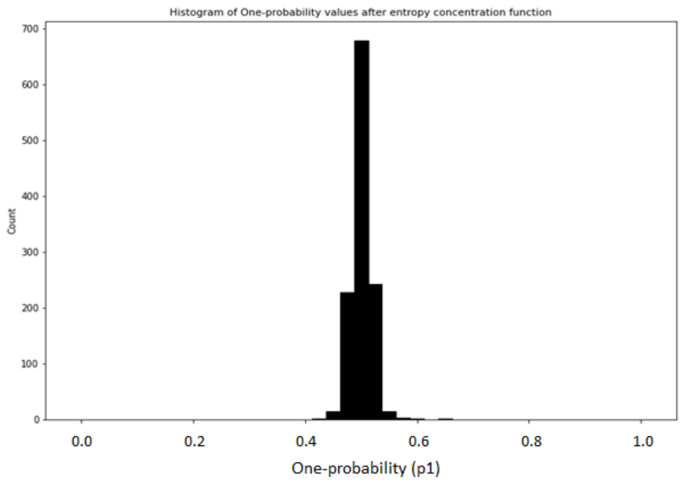

Figure 8: One-probability distribution (histogram) of SRAM PUF data after applying the entropy concentration function based on a matrix multiplication with a RM(5,8) parity check matrix.

We consider the SRAM PUF based generated noise samples to be non-i.i.d., meaning not identically and independently distributed because of the following reasons:

- The behavior of the individual SRAM PUF cells is by its nature non-identical, since each cell has its own individual distribution described by its one-probability,
- The entropy concentrating transform generates noise samples from SRAM PUF cell evaluations in a manner which is not fully independent.

For this reason, we run the non-i.i.d. entropy assessment flow of the NIST-SP800-90B test suite [22] on the sequential dataset. As a result we find an estimated entropy of 0.84 bits per sample.

We run the entropy assessment restart test [22] on our restart dataset of 1000 times 1000 samples, which are used in the test as a 1000 by 1000 matrix of output samples (bits). As additional input we need to provide an initial entropy estimate, which is the minimum of our model-based submitter entropy and the entropy estimate created by the non-i.i.d. entropy assessment. In our case, the submitter entropy was the lowest and hence the value of 0.37 bits per sample is used as initial entropy estimate input. The restart tests verify the following:

1. A restart sanity check: check that the frequency of the most common value in the rows and the columns of the matrix is not significantly larger than the expected value, given the initial entropy estimate. This sanity check passes with our restart dataset.
2. A restart validation check: check that the entropy estimates over the columns and over the rows of the matrix is at least half of the initial entropy estimate. This validation check passes as the test suite estimates

a row entropy of 0.81 bits per sample and a column entropy of 0.84 bits per sample, which are both higher than half of the submitter entropy of 0.37 bits per sample.

With all tests passing we find a minimum noise entropy estimate of 0.37 bits per sample. This is the value that we need to consider when feeding noise into the DRBG on this platform. It means in practice that for providing 256 bits of noise entropy into our DRBG, we need to take at least 256/0.37=692 bits of entropy source output data. Given the efficiency of our entropy concentration function, this requires at least 692/37*256 ≈ 4787 bits (~600 Bytes) of SRAM PUF data.

It should be noted that the above entropy assessment serves merely as an example. In a proper entropy source validation run, we would need to repeat the analysis over multiple devices and focus on the worst-case SRAM PUF noise condition, which is typically at the lowest operating temperature. This will result in an implementation that uses slightly more SRAM PUF data than the estimate provided in the above example. Nonetheless, the example shows that constructing an SRAM PUF based entropy source that fulfills the NIST SP800-90 criteria is feasible.

V. Conclusions

In this paper we have shown how a NIST SP800-90 compliant random number generator can be constructed with an entropy source that is based on using uninitialized SRAM memory. Even though the NIST specifications seem to be written with a continuous or "temporal" noise source model in mind, we have shown how to apply the specifications to our "spatial" SRAM PUF noise source. In this spatial model, the entropy is not released over time but instead generated at one specific moment in time at power up of the SRAM. The number of samples that can be produced per power-up is limited by the SRAM size, which is a property that needs to be taken into account by the implementation of the random number generator. Another specific aspect of using the SRAM PUF as entropy noise source is to deal with the sparse and diluted characteristics of the SRAM PUF noise. An entropy concentration function has been proposed as part of the digitization step to improve the properties of the Raw Data output. With this addition we have shown that our SRAM PUF based noise source can successfully pass the NIST entropy validation tests on an actual microcontroller device.

With the construction presented in this paper, we have shown how SRAM PUF technology can be leveraged to instantiate a strong RNG on almost any microcontroller, hence providing a universal solution for strong randomness in the Internet of Things. Intrinsic ID has created an embedded software product called Zign RNG [19] around this proposition. It is delivered as a compiled library for a specific CPU and comes with documentation including datasheet, API reference manual and entropy source validation guide for use in FIPS certification projects. The implemented cryptographic algorithms in the DRBG have been certified using the NIST CAVP program [26].

References

[1] Dan Petro - Bishop Fox, Blog August 05, 2021: "You're Doing IoT RNG", https://bishopfox.com/blog/youre-doing-iot-rng

[2] IoT Analytics – "State of IoT 2021: Number of connected IoT devices growing 9% to 12.3 billion globally, cellular IoT now surpassing 2 billion", https://iot-analytics.com/number-connected-iot-devices/

[3] GSA white paper: "Preventing a $500 Attack Destroying your IoT Devices" https://www.intrinsic-id.com/resources/white-papers/landing-page-white-paper-preventing-a-500-attack/

[4] J.P. Aumasson, "Serious cryptography: a practical introduction to modern encryption". No Starch Press, Inc., 2018.

[5] Intrinsic ID Whitepaper, "Protecting the IoT with Invisible Keys IoT Security with Unclonable Identities", https://www.intrinsic-id.com/resources/white-papers/protecting-iot-invisible-keys-white-paper/ .

[6] Intrinsic ID Whitepaper, "SRAM PUF: The Secure Silicon Fingerprint", https://www.intrinsic-id.com/resources/white-papers/white-paper-sram-puf-secure-silicon-fingerprint/

[7] G.J. Schrijen, C. Garlati, "Physical Unclonable Functions to the Rescue, A new way to establish trust in silicon", Embedded World 2018.

[8] ArsTechnica – "PS3 hacked through poor cryptography implementation", https://arstechnica.com/gaming/2010/12/ps3-hacked-through-poor-implementation-of-cryptography/

[9] B.Buchanan on Medium.com, "Not Playing Randomly: The Sony PS3 and Bitcoin Crypto Hacks", https://medium.com/asecuritysite-when-bob-met-alice/not-playing-randomly-the-sony-ps3-and-bitcoin-crypto-hacks-c1fe92bea9bc

[10] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman, "Mining your Ps and Qs: detection of widespread weak keys in network device," Proceedings of the 21st USENIX Security Symposium, August 2012.

[11] Ecommerce Times article, "Android Flaw Could Empty Bitcoin Wallets", https://www.ecommercetimes.com/story/android-flaw-could-empty-bitcoin-wallets-78702.html

[12] NIST SP 800-90A Rev.1, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators", https://csrc.nist.gov/publications/detail/sp/800-90a/rev-1/final

[13] NIST SP 800-90B, "Recommendation for the Entropy Sources Used for Random Bit Generation", https://csrc.nist.gov/publications/detail/sp/800-90b/final

[14] NIST SP 800-90C, "Recommendation for Random Bit Generator (RBG) Constructions", https://csrc.nist.gov/publications/detail/sp/800-90c/draft

[15] FIPS 140-3, "Security Requirements for Cryptographic Modules", https://csrc.nist.gov/publications/detail/fips/140/3/final

[16] Y. Dodis, L. Reyzin, A. Smith – "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data", In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT'04. LNCS, vol. 3027, pp. 523–540. Springer-Verlag, Heidelberg (2004)

[17] C. Bösch, J. Guajardo, A.R. Sadeghi, J. Shokrollahi, P. Tuyls – "Efficient helperdata key extractor on FPGAs. In: Oswald, E., Rohatgi, P. (eds.) CHES'08. LNCS, vol. 5154, pp. 181–197. Springer-Verlag, Heidelberg (2008)

[18] V. van der Leest, E. van der Sluis, B. Preneel – "Soft Decision Error Correction for Compact Memory-Based PUFs using a Single Enrollment", CHES 2012.

[19] Intrinsic ID – Zign RNG product, https://www.intrinsic-id.com/products/zign-rng/

[20] NIST Cryptographic Algorithm Validation Program (CAVP), https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program

[21] Roel Maes – "An accurate probabilistic reliability model for silicon PUFs", CHES 2013, https://eprint.iacr.org/2013/376.pdf

[22] NIST SP800-90B Entropy Assessment, https://github.com/usnistgov/SP800-90B_EntropyAssessment

[23] Katzenbeisser, S., Kocabaş, Ü., Rožić, V., Sadeghi, A. R., Verbauwhede, I., & Wachsmann, C. (2012). PUFs: Myth, fact or busted? A security evaluation of Physically Unclonable Functions (PUFs) cast in silicon. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7428 LNCS*, 283–301. https://doi.org/10.1007/978-3-642-33027-8_17

[24] Schrijen, G. J., & Van Der Leest, V. (2012). Comparative analysis of SRAM memories used as PUF primitives. *Proceedings -Design, Automation and Test in Europe, DATE*, 1319–1324. https://doi.org/10.1109/date.2012.6176696

[25] Claes, M., Van Der Leest, V., & Braeken, A. (2012). Comparison of SRAM and FF PUF in 65nm technology. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 7161 LNCS, 47–64. https://doi.org/10.1007/978-3-642-29615-4_5

[26] NIST CAVP certification results for Intrinsic ID's Zign RNG product, validation number A1993, https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=14480