Zign RNG: Using Standard SRAM to Create a Universal Approach for Adding Strong Cryptographic Randomness to Any IoT Device

A poor RNG generates predictable keys, directly reducing the security level of a cryptographic mechanism.

Security Threats Grow Along with the IoT

It is estimated that by 2025, there will be more than 27 billion devices connected to the Internet.[1] Interconnecting billions of devices on the internet of things (IoT), has exposed the world to a plethora of new security-related threats. 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.

Cryptography – a field dedicated to keeping information secret and safe by transforming it into form that unintended recipients cannot understand – offers several important tools to guard against these increasing attacks. Many cryptographic protocols depend on the availability of unpredictable random numbers. For example, they are used in encryption schemes as initialization vectors, for key-establishment protocols, and for the generation of secret keys, PINs, and passwords.

A poor random-number generator generates predictable keys, directly reducing the security level of a cryptographic mechanism using such keys and giving an undesirable advantage to an attacker. Over the past years there have been several examples where poor-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.[2,3] Insufficient randomness in the application of ECDSA signatures was found in 2013 in

[1] 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/

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

[3] 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

Android implementations of bit-coin wallets.[4] In 2012 researchers discovered issues with the RSA keys of thousands of Internet-connected devices.[5] More recently, a study from security company Bishop Fox showed issues with the use of random generators in IoT devices which put billions of devices at risk.[6]

Strong cryptographic randomness is essential to the quality of protection provided by any cryptographic system that relies on random numbers

Strong Cryptographic Randomness for Every Chip

Fortunately, there is an approach that addresses the need for true random-number generation on embedded devices. This approach employs SRAM-based Physical Unclonable Functions (PUFs), using the intrinsically random properties of SRAM start-up values, which can be made available on any device to serve as a true-random source of entropy for cryptographic functions.

Intrinsic ID has created an embedded software product called Zign® RNG[7] following the NIST SP 800-90 standards.[8,9,10] Following these standards is a requirement for devices that claim FIPS 140-3 compliance.[11] 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 deterministic random bit generator (DRBG) employed in Zign RNG have been certified using the NIST CAVP program.[12]

This backgrounder focuses on how Zign RNG uses SRAM PUFs to build a strong cryptographic random number generator on embedded devices. First, we summarize the properties of SRAM PUFs and explain how they can be used as a source of true randomness. Then we detail how Zign RNG uses an SRAM PUF source to build a FIPS-compliant random generator according to the NIST SP 800-90 standards.

---

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

[5] 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.

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

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

[8] 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

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

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

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

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

SRAM PUF responses can be used as a source of non-repeating true randomness.

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 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 in cryptography. However, SRAM PUF behavior is 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.

SRAM PUF Behavior

An SRAM memory consists of an array of SRAM cells. Each SRAM cell consists of two cross-coupled inverters, each built up by a p- and an 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.’

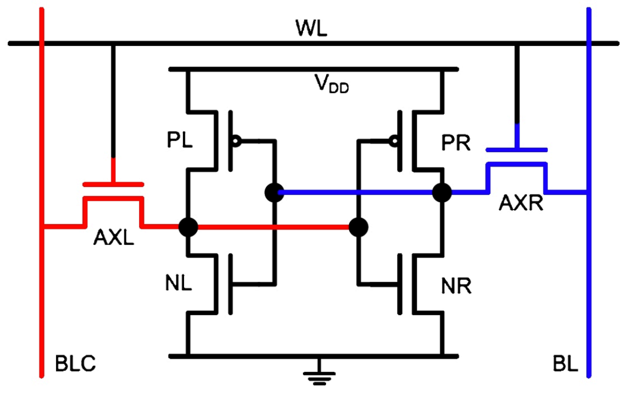

Figure 1 – Schematic of 6-transitor SRAM cell. The left inverter consists of p-MOS transistor PL and n-MOS transistor NL and is cross-coupled with right inverter consisting of transistors PR and NR. Transistors AXL and AXR are access transistors for read and write operations. The SRAM cell is accessed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

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

A small fraction of the bit cells will 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, producing noisy results at every power-up. SRAM PUF responses can be used as a source of non-repeating true randomness, using this small percentage of noisy cells.

Randomness Generation Based on SRAM PUFs

The predictability of a PUF-response bit can be quantified using a measure 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 2, most SRAM PUF cells have a one-probability value either close to 0.0 or close to 1.0, indicating either a stable 0-producing cell or a stable 1-producing cell, respectively. Only a relatively few cells will have a one-probability not close to either 0.0 or 1.0, indicating an unstable cell with a power-up response value that is somewhat unpredictable. A small (but non-negligible) minority of cells will have a one-probability close to 0.5, which indicates a fully unpredictable response behavior. These unstable cells generate fresh entropy – also called noise entropy – upon every power-up of the SRAM memory.

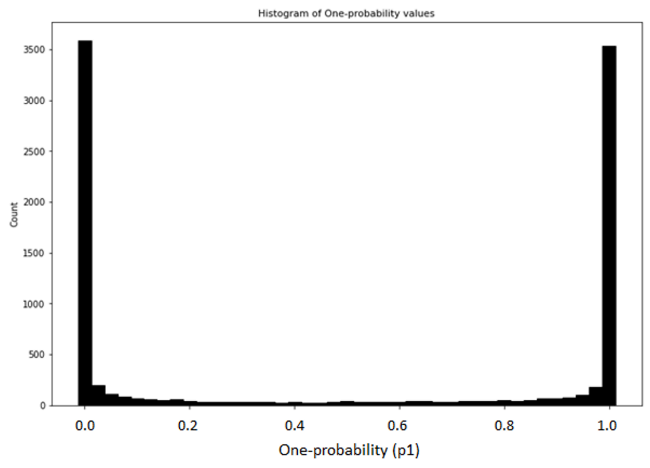

Figure 2 – One-probability distribution of a 1KB SRAM array measured 1000 times at room temperature. Most of the bit cells power up as a very stable digital 0 or 1 value (resulting in the peaks in the above histogram). A minority of bit cells has a more random power-up behavior and contribute to the noise entropy.

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. This noise entropy provides the basis for using the SRAM PUF as an entropy source in a FIPS-compliant random number generator.

The noise entropy contained in the typical one-probabilities as observed in SRAM PUF responses is both sparse and diluted:

- The total noise entropy produced by an SRAM PUF array will be generated mostly 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 be rather low, or in other words the noise entropy in an SRAM PUF response is ***diluted***.

Because of these properties, an additional entropy-concentration function on the SRAM PUF output is required to fit the NIST entropy-source model, which is described in the next section.

Zign RNG uses SRAM PUFs to create an approved entropy source according to the NIST SP 800-90B specification.

FIPS-Compliant Randomness Generation

FIPS 140-3-compliant security modules need to have a random-number generator that is compliant to the NIST SP 800-90 specifications.[8,9,10] According to these specifications, an approved random-number generator consists of a DRBG that is requesting entropy from a randomness source such as a NIST-approved entropy source, as shown in Figure 3.

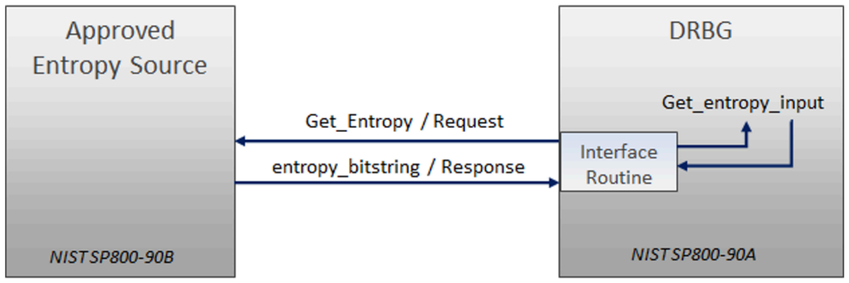

Figure 3 – Schematic architecture of an approved random-number generator construction according to the NIST SP 800-90C specification.[10]

Zign RNG uses SRAM PUFs to create an approved entropy source according to the NIST SP 800-90B recommendations, with the addition of an entropy concentration function.

Entropy Source

Guidelines for NIST-approved entropy sources are given in the NIST SP 800-90B specification.[9] According to this specification, an entropy source should include the components as depicted in Figure 4. These components include a digital noise source, whose output is raw data that is 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.

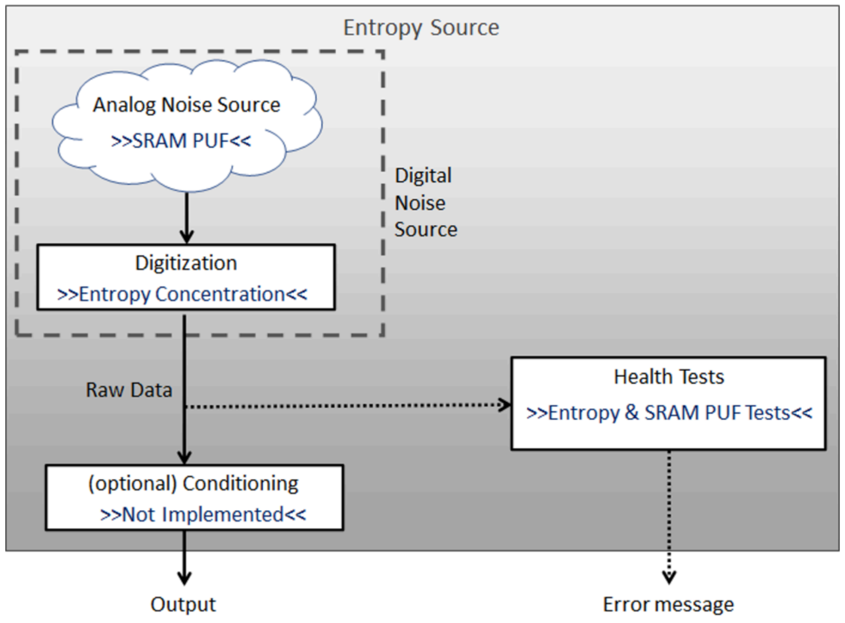

Figure 4 – Entropy Source Model according to NIST SP 800-90B specification.[9] Texts between >>marks<< indicate the implementations used in Zign RNG.

Analog Noise Source

To create an SRAM PUF-based entropy source, Zign RNG uses 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 used 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.

Digitization

The digitization step is needed because the noise entropy present in the binary PUF response is sparse and diluted. The NIST SP 800-90B 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 has sufficient entropy for the intended application. This is why using SRAM PUF bits directly as noise samples is suboptimal because many bit locations do not contain any noise entropy at all (due to the sparseness).

By implementing an entropy-concentration function as a "digitization" step, Zign RNG transforms the sparse and diluted noise entropy in the SRAM PUF bits into a smaller set of output bits in which the noise entropy is concentrated. However, it is important that the entropy-concentration function doesn't obfuscate the statistical properties of the physical behavior on which the noise source relies as the origin of the noise entropy. This requirement rules out the use of, for example, 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.

Instead, Zign RNG employs a simple binary matrix multiplication. Such a matrix can be designed so that 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 that largely retains the statistical properties of the noise source, enabling meaningful testing and validation of the noise-source samples. Zign RNG uses the parity-check matrix of a Reed-Muller code, which is well-suited to this purpose.

Health Tests

The output of the digital noise source in Figure 4, the raw data, must be validated by means of health tests designed 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. 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 requires that both startup tests and continuous tests are included.[9] Whereas startup tests run after powering up or rebooting to verify that the noise source components are operational, continuous tests run when the noise source is operating to detect output failures. An SRAM PUF-based random source does not produce random outputs continuously, but instead delivers entropy only at power-up of the SRAM memory. So, this is the moment at which Zign RNG runs the "continuous" tests as described by NIST.

NIST SP 800-90B mandates two statistical tests on the raw data:

- Repetition Count test: tests for too-long sequences of repeating noise sample values, and
- 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 that are not sufficiently covered by the two mandatory tests.

For its SRAM PUF-based noise source, Intrinsic ID Zign RNG adds tests on the actual PUF-response values before the digitization step to increase the detection sensitivity. The Repetition Count and Adaptive Proportion tests (with adapted parameters) are run on PUF-response values directly.

Additionally, Zign RNG performs an SRAM state test to detect reuse of SRAM PUF values without proper re-powering in between (a very specific SRAM PUF failure mode). 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, a check is performed to confirm that 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.

Conditioning

The conditioning function shown in Figure 4 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 that cryptographically compresses the entropy input upon instantiation, a conditioning function in the entropy source is generally not needed, so this is not included in Zign RNG.

Interfaces

A NIST-approved entropy source should include, at a minimum, the following conceptual interface functions: GetEntropy, GetNoise, and HealthTest. Intrinsic ID Zign RNG implements these interface functions using SRAM PUF response values as follows:

- **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), Zign RNG passes the PUF response values through the digitization function. After applying the health tests as described earlier, Zign RNG provides the raw data 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 should not be callable in a regular operational mode. In Zign RNG, 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 (it does not output any random data). In Zign RNG, 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).

Zign RNG implements a DRBG as specified in the NIST SP 800-90A specification.

DRBG

The deterministic random bit generator (DRBG) is the main mechanism that is called by the application to deliver random data. Approved cryptographic mechanisms, which are not specific to the noise source, are described in the NIST SP 800-90A specification.[8] The functional model for a DRBG as described by NIST is depicted in Figure 5.

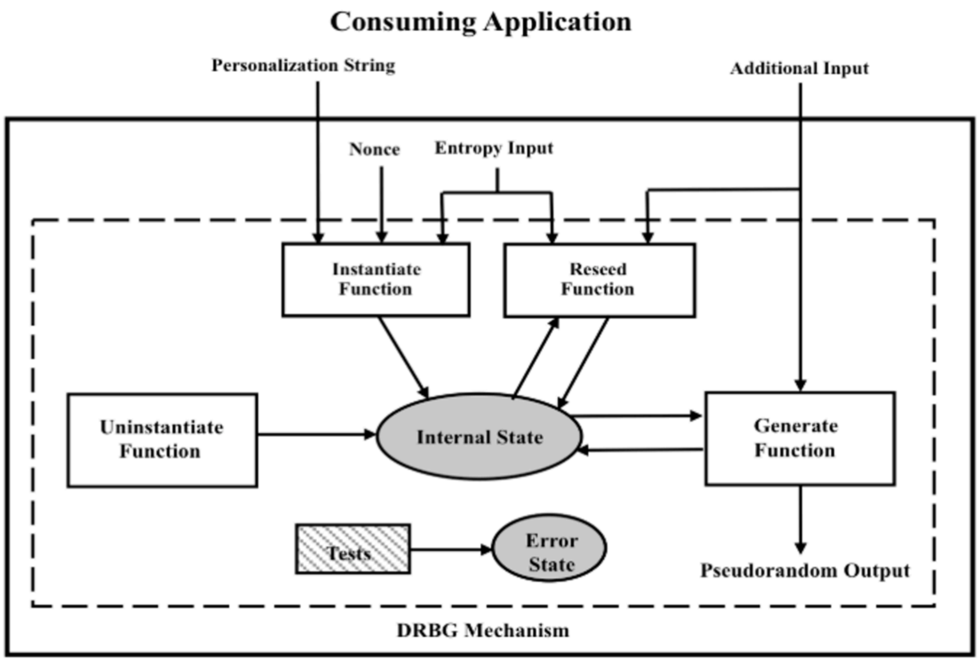

Figure 5 – DRBG Functional Model, as taken from the NIST SP 800-90A specification.[8]

An application requests random bits by calling the "generate" function. Application-specific input data can be mixed in via an optional additional input. 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 a pseudo-random function is validated and certified via the NIST CAVP program.[13] The implementation used by Zign RNG has received this certification.

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

Before calling 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 it with an optional personalization string input provided by the application.

The NIST specification requires an "uninstantiate" function to destroy the internal state in case of certain security-related events, such as detected breaches. The specification also requires 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 provide additional entropy to 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 is no longer guaranteed. 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 an SHA or AES based pseudo-random function. An internal counter keeps track of the number of requests that have been handled and the DRBG will indicate that a reseed is required when the maximum number of requests has been reached.

Special Considerations

For SRAM PUF-based entropy sources, there are two special considerations:

1. After the DRBG has been uninstantiated, a repower of the SRAM (or the device) is needed to generate fresh noise entropy for the noise source before a new instantiation can take place.
2. A reseed can only be implemented from the SRAM PUF-based noise source when the SRAM memory has a dedicated SRAM power switch. If the SRAM is powered along with the rest of the device, a reseed cannot be implemented from the same entropy source. However, this reseed limitation is not an issue in actual practice, as typically multiple random bytes are included in a single request and 2^{48} requests can be served without reseeding. This huge number of requests would not be reached even if the DRBG were called once every millisecond for 100 years.

Entropy Source Validation

The entropy source in Zign RNG has been validated according to the NIST SP 800-90B specification. Details of this validation are available in the technical paper, "Creating an Efficient Random Number Generator Using Standard SRAM." Zign RNG passed all required tests.[14]

[14] Geert-Jan Schrijen en Roel Maes, "Creating an Efficient Random Number Generator Using Standard SRAM", https://www.intrinsic-id.com/wp-content/uploads/2022/07/SecureHardware_schrijen_paper.pdf

Zign RNG provides a universal solution for strong randomness in the IoT.

Conclusion

The IoT is growing at a rapid rate and is projected to include 27 billion devices by 2025. With this growth, security threats multiply as well. Random-number generators are an important component of cryptographic security systems. However, in recent years, there have been multiple examples of wide-spread security issues with existing random-number generation systems. The NIST SP 800-90 specification seeks to avoid security breaches by laying out the necessary components, functions, validations for secure random-number generation, and provides a path for validation of candidate systems.

Intrinsic ID has created an embedded software product called Zign RNG which uses SRAM PUF response data to create a source of entropy that can be used on any chip. The implemented cryptographic algorithms in the DRBG have been certified using the NIST CAVP program.[12]

This technical backgrounder has detailed how Zign RNG uses uninitialized SRAM memory to provide a NIST SP 800-90-compliant random number generator constructed with an entropy source that is based on standard components available in virtually any device, providing a universal solution for strong randomness in the IoT.