CESEL: Securing a Mote for 20 Years

Kevin Kiningham, Mark Horowitz, Philip Levis, and Dan Boneh
Stanford University

Abstract

Embedded wireless sensors, once deployed, may remain in active use for decades. At the same time, as motes come to dominate both the number of hosts and data traffic of the Internet, their security will become fundamental to general Internet security. This paper argues that the next generation of embedded networked sensor devices ("motes") should consider this tension in their basic design and be designed to remain secure for 20 years in a rapidly changing and evolving security and cryptographic landscape.

The key insight in this paper is that the economics of modern system-on-a-chip (SoC) designs provides ample space for hardware accelerators and cryptographic engines. A next generation mote can therefore include many such co-processors and features at almost no production cost. The paper describes an initial design for what hardware security support such a device should have, focusing on five hardware primitives: an atomic, unique counter, a random number generator based on physical entropy, additional instructions to accelerate symmetric ciphers, an elliptic curve accelerator, and support for modular polynomial multiplication used in post-quantum cryptographic signing algorithms. We call this architecture CESEL.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*Real-time and embedded systems*

General Terms

Hardware Acceleration, Low Power Cryptography

Keywords

Flexible Hardware, Internet of Things, Long Term Security

1 Introduction

Computer and network security has seen many transformations in the past 20 years. In 1996, SSL, used in HTTPS (secure HTTP) was only beginning to be defined: it was vulnerable to many attacks and is considered insecure today. Telnet was still a common login option on shared machines. NIST hadn't yet announced it was seeking a successor to the DES ciphersuite: AES, the standard cipher relied on today, wasn't standardized until 2000. Since 1996, we have seen trojan horses, worms, viruses, DNS cache poisoning, cross-site-scripting, phishing, botnets, ransomware, advanced persistent threats, and state-sponsored cyberattacks.

At the same time, we have seen the growth of embedded systems and sensor networks[3]. Recent technological shifts in wireless protocol support and microcontrollers are making it possible to build long lived personal body monitors, smart homes, smart cars, and smart cities. These embedded devices, once deployed, may remain deployed for many years. A smart lock, for example, can't be changed every year, nor can a smart window or a smart medical implant. Smart, embedded sensing devices, once deployed, will often last as long as 20 years. These devices, once connected, present a potential security vulnerability. A smart lock can be unlocked, a smart car could cause a collision or kidnapping, a smart medical implant could kill you. All have tremendous security implications.

We argue that the next generation of motes must address the fact that they will need to remain secure over their entire lifetime. We use 20 years as a rule of thumb for what "entire lifetime" means; while many appliances, cars, and other devices will be retired well before 20 years, some will be held onto for a very long time. Sensor network conferences such as EWSN and SenSys are 12-13 years old, and papers published in 2015 use hardware developed in 2003[15][12]. If even a small research community uses decade-old hardware, it is not far fetched to think that consumers will hold onto it for 20 years.

This paper identifies five primitives that would greatly increase the chances that a device can remain secure for 20 years and describes a system architecture that includes them, named CESEL. These five primitives are:

- an atomic, non-volatile counter to generate unique nonces,

- a random number generator with entropy guarantees,
- instructions to accelerate symmetric block ciphers,
- a co-processor supporting elliptic curve point operations, and
- a co-processor supporting modular polynomial arithmetic.

Section 2 motivates the need for 20 year security. Section 3 describes five key requirements the next generation of motes should have. Section 4 proposes several extensions to existing hardware that would enable a device to support ciphers and remain secure for the next 20 years. Section 5 concludes.

2 Securing for 20 Years

As the number of wireless sensors grow, it becomes increasingly difficult to physically upgrade or replace each sensor. Sensors can be placed in physically hard to reach places like the human body or remote locations. In networks with thousands or millions of sensors, it becomes impractical to replace each sensor's hardware without a massive amount of effort. Because of this, it is likely that sensors designed today will need to be secured and usable for many years or decades to come.

However, securing communication between sensor nodes present a significant engineering challenge when designing sensors to last multiple decades. Several cryptosystems have went from recommended best practice to completely insecure in less than 20 years (see Table 1). In addition, the most common public-key algorithms have known weaknesses to quantum computers, which may be widely available in as little as a decade[6]. Since current cryptographic methods can become outdated and insecure, the long term security of sensor communication will rely on its ability to adapt to future ciphers and crypto.

While there is strong evidence on the mathematical basis of future ciphers and cryptosystems, the exact form they will take is still unknown. For example, new ciphers are designed and proposed quite often, but many rely on basic structures and primitives like feistel-networks or add-rotate-xor (ARX). Therefore, rather than define a particular algorithm, a next generation mote should design its cryptographic accelerators to be programmable and flexible.

Table 1. Lifetimes of common cryptographic algorithms

	Algorithm	Release	Insecure	Lifetime
Symmetric Ciphers	RC2[13]	1996	1997	1 year
	RC4[4]	1994	2013	19 years
	DES[16]	1979	1994	16 years
	3DES[4]	1998	2015	17 years
	AES	1998	-	>17 years
	Camellia	2000	-	>15 years
Hash Functions	MD5[17]	1992	2004	12 years
	SHA-1[17]	1995	2004	9 years
	SHA-256	2000	-	>15 years

However, many wireless sensors have limited computational and memory resources and strong cryptography is often too expensive for embedded systems to implement in software. As a result, many manufactures have added hardware accelerators to reduce the power used during encryption and decryption. However, these accelerators typically target a single cryptographic algorithm (most commonly AES), and cannot be used for newer cryptographic standards. Additionally, cryptographic operations that are not accelerated (such as public-key cryptography) end up not being used, leaving communication insecure. CESEL adds flexible, hardware based, cryptographic accelerators, allowing for fast and secure cryptography.

Including these cryptographic accelerators into next generation mote SoCs would also have minimal additional cost. While they would increase design and testing costs, once in silicon their costs would be minimal. In modern SoCs, the cost of enough silicon for millions of gates is tiny, so the incremental cost of adding a few hundred thousand more is very small; the Atmel SAM4L has enough area for tens of millions of gates[20, pp.-3], and costs just $3.40 on Digi-Key. We argue this fractional increase in cost is well worth cryptographically protecting against the future.

Table 2. State-of-the-art mote microcontrollers

Microcontroller	Flash	SRAM	Core Speed	Package	Hw. AES
NRF51822	256 KB	16 KB	16 MHz	6x6 mm	YES
QN902X	128 KB	64 KB	32 MHz	6x6 mm	YES
CC2538	512 KB	16 KB	32 MHz	8x8 mm	YES
STM32L0x1	32 KB	8 KB	32 MHz	7x7 mm	YES
MKW40Z	160 KB	20 KB	48 MHz	7x7 mm	YES

3 Security Requirements

CESEL's design focuses on what cryptographic primitives and mechanisms a mote should have. The design assumes that a mote's software and hardware are secure and that application code can trust its own storage and code. CESEL therefore does not consider attacks such as physical compromise or power analysis; these relate to the physical construction of the SoC, and in this study we constrain ourselves to an architectural design. Similarly, the design does not consider software security threats, such as buffer overflows or resource exhaustion: these are best handled by an operating system and programming language.

Today, mote security is typically bootstrapped by installing a shared private key on every mote in a network[23]. This key provides the necessary secret to provide link-layer secrecy, integrity, and authentication. This approach is tolerable for stovepipe designs and single-vendor networks, but as applications become richer, longer-lived, and more complex, we assume that a more dynamic and powerful authentication system will sometimes be needed: certificates and signatures. Certificates and signatures would allow mote networks to dynamically authenticate nodes without having to share a private key. For example, suppose that a smart door lock wants to authenticate with a home network and establish credentials to be able to trigger the alarm system. The lock first needs to authenticate the alarm system, to protect against spoofing attacks: it needs to verify a signature from the alarm system as well as the system's certificate chain. Requiring that the lock know a shared key that is shared across many product lines, in contrast, is vulnerable to anyone learning what this shared key is: a single leaked source tree, or a single compromised device,

would open the system to attack.

We therefore take a long-term view of what cryptographic primitives motes may need to use in the future. They will not only need to support symmetric ciphers that may evolve and change, but also public key cryptosystems. Furthermore, there are lower-level primitives, such as nonces and random numbers, that many protocols depend on. Taking a long-term view, we argue that next generation motes should include five architectural building blocks:

- an atomic, non-volatile counter to generate unique nonces,
- a random number generator with entropy guarantees,
- instructions to accelerate symmetric block ciphers,
- a co-processor supporting elliptic curve point operations, and
- a co-processor supporting modular polynomial arithmetic, in case quantum computers become a reality.

3.1 Random Number and Nonce Generation

Random number generation is a key piece of modern cryptography and security[9]. A fast, secure source for random numbers is used in key generation, securing networking protocols, and even high level application security like ASLR. Misuse of random numbers has lead to a large number of security bugs[9].

In embedded systems, it is particularly challenging to find secure sources of entropy, and the sources that are available are often of low quality, or can be easily compromised. For example, many embedded systems use external sources, like ADC readings, or difficult to guess timing information, like the current cycle count. However, if an attacker gains physical control of a device they can force any external source to take on whatever value they like. Additionally, when a device first boots there is a limited source of timing information[2], which means applications must choose between waiting for enough entropy to be gathered or using a low quality entropy source. CESEL eliminates this problem by adding a dedicated random number generator based on a physical process like thermal noise, allowing for fast and secure random number generation.

Additionally, unique nonce generation is another crucial primitive needed for many cryptographic algorithms and networking protocols. For example, nonces are required in CBC-mode for block ciphers to ensure that identical plaintexts are not encrypted to the same ciphertext. If a nonce is reused, information is leaked about the plaintext to an attacker, which violates ciphertext indistinguishably. This exact method was used to decrypt TLS traffic in the BEAST attack[1] as well as break the 802.11 encryption standard, WEP[7].

Unfortunately, it can be quite difficult to securely generate unique nonces. Applications must be sure to prevent reuse even in the case of power failure or software bugs. However, ensuring each nonce is unique can be done easily in hardware. For this reason, CESEL includes a dedicated nonce generator that can be used regardless of power or software failures.

3.2 Symmetric Ciphers

Since the connection between sensor nodes is untrusted, any data sent between nodes may be intercepted by an attacker. An attacker may then inject, modify, or delete any packets they desire. This means that sensitive communications, like sensor readings or cryptographic keys, must be kept confidential and all packets must be checked for integrity and authenticity.

In CESEL, this requirement is met by accelerating authenticated symmetric cyphers. Symmetric ciphers encrypt the data with a shared secret key, which prevents an attacker from reading the data. For authentication, block ciphers can be run in a mode of operation that allows for data authentication. CESEL accelerates both the block ciphers and common modes of operation, allowing for efficient confidentiality, authenticity, and integrity over an untrusted connection.

3.3 Public Key Cryptography

Public-key cryptography can also be used to guarantee data authenticity and integrity. The sender produces an unforgeable signature over the data using their private key. The recipient can then verify the sender's signature using the sender's public key. Note that this method does not rely on the recipient knowing the sender's private key. This prevents a malicious recipient from forging packets on behalf of the original sender.

For each of data confidentiality, authentication, and integrity, we rely on the fact that each sensor is able to generate and store a cryptographic key that it can use to communicate with other devices. This may happen when a sensor node is added to a new network and needs to transmit data to a device it hasn't seen before. A key agreement protocol allows two devices that have not communicated before to derive a common key over an untrusted link.

Unlike confidentiality, authentication, and integrity, it is impossible for key agreement to be implemented using only symmetric-key cryptography if there is not already a secure, shared secret between parties. Instead, most common key agreement protocols, such as Elliptic Curve Diffie-Hellman Exchange (ECDHE), use methods based on the primitives in public-key cryptography to derive a shared secret key.

However, public-key operations can be extremely slow on embedded devices[27]. Thus, public-key cryptography is frequently considered to be too expensive for embedded devices, requiring wireless sensors to rely on less secure methods for authentication and key exchange. For example, Bluetooth LE relies on an custom and insecure[25] key-exchange protocol, in order to avoid the computational expense of public-key cryptography.

CESEL accelerates public-key operations by providing efficient support for modular arithmetic, including addition and multiplication, allowing devices to securely authenticate and communicate keys.

4 Proposed Design

CESEL extends the typical architecture of an SoC with five additional elements: an atomic, non-volatile hardware counter, a hardware based random number

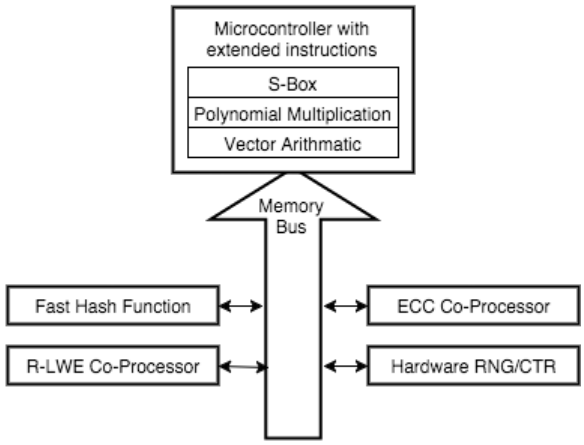

Figure 1. CESEL Architecture

generator, additional instructions to accelerate symmetric block ciphers, a co-processor that efficiently supports elliptic curve operations, and a co-processor to support post-quantum public-key operations.

Each co-processor communicates with the main processor via the memory bus, allowing them to perform independent computation and preventing the CPU from stalling on long co-processor operations. Additionally, decoupling the processor from each co-processor allows each to be independently power-gated. Since the cryptographic co-processors are only expected to be operating for a small portion of time, this reduces leakage power when the co-processor is not in-use.

4.1 Hardware Random Number Generator and Non-Volatile Atomic Counter

CESEL includes a hardware based random number generator as a core part of its architecture. Software random number generators sample an external source of entropy (such as an unconnected ADC pin) and used this source as an input to a cryptographically secure pseudo-random number generator (CSPRNG). However, sampling an ADC or other external entropy source can be slow, which makes it difficult to accumulate enough entropy for repeated cryptographic operations. Additionally, if the embedded device is physically controlled by an attacker, the attacker will then also control all external entropy sources. They can then use this control to predict the resulting generator output.

On the other hand, a hardware random number generator directly samples a fast internal source of entropy such as a metastable circuit[8]. A metastable circuit operates by rapidly amplifying inherent thermal noise, resulting in a highly unpredictable output each time it is sampled. The resulting bitstream is then fed into a CSPRNG, reducing the raw entropy source into a *conditioned* or unbiased output. A hardware random number generator thus has the advantage of being both faster and higher-quality than a software only solution.

CESEL also implements a hardware non-volatile atomic counter. Nonces are used in many cryptographic algorithms and reusing the same nonce more than once can prove catastrophic for security. If power to the device fails at the wrong time or there is a bug present in the application, it is quite easy for a sensor node to accidentally reuse a nonce, which can lead to leaking secret data or cryptographic keys. A hardware counter would be guaranteed to return a unique value every time it is queried, regardless of power failure or other processor state.

Atomic increments of non-volatile memory are easy to implement in hardware and exist in many real world systems. A typical implementation consists of two non-volatile counter registers and a non-volatile bit flag. The bit flag encodes which register contains the current count and is considered *active*. On each atomic increment, the non-active register is erased and then loaded with the value of the active register plus one. The bit flag is then flipped, and the non-active register becomes the active register and visa-versa.

Importantly, we only return the new value after the increment has completed successfully. Thus, any code that relies on a unique value being returned cannot complete until the counter has successfully incremented, and the counter can never reset, even in the case of power failure. This guarantee significantly improves data security by making it impossible for an application to reuse a nonce.

4.2 Symmetric Cipher Acceleration

CESEL uses authenticated symmetric ciphers to guarantee confidentiality, authentication, and integrity on data transmitted over an insecure channel between nodes. There are two basic types of commonly used symmetric key ciphers: block ciphers and stream ciphers. Stream ciphers encrypt plaintext messages one letter (typically a single bit) at at time. Block ciphers operate on a fixed size group of bits called a block and typically perform a fixed number of repeated operations on the block called rounds. In order to encrypt more than a single block of data, block ciphers must be paired with a mode of operation, such as counter (CTR) or cipher block chaining (CBC). Additionally, some modes of operation, such as offset codebook (OCB) or Galois/Counter Mode (GCM), are able authenticate encrypted data for little additional cost.

CESEL extends a normal sensor node's CPU with three groups of instructions to accelerate symmetric ciphers. The first instruction is the *S-box* or substitution instruction. The instruction takes a vector of eight bytes and uses each byte as an offset into a rewritable 256-byte substitution table. Each byte is then replaced with it's given value in the substitution table. Many symmetric ciphers use S-boxes as a basic primitive in their operation, and the time it takes for to perform this lookup can frequently dominate cipher performance.

The second instruction is the *polynomial multiplication* instruction. The polynomial multiplication instruction performs a carryless multiplication of two vectors of eight bytes in a Galois Field $GF(2^n)$, where n is between 1 and 128. Polynomial multiplication is used in several different symmetric ciphers (such as AES) and MAC constructions[10] (such as the GCM mode of operation). Polynomial multiplication is also very slow to implement in

software but quite fast to implement in hardware due to many fixed sized shifts and exclusive or operations.

The final three instructions are *vector add*, *vector xor*, and *vector shift by constant*. The first two instructions, vector add and vector xor, take two vectors of four 32-bit values and performs an element-wise add or exclusive or respectively. Vector shift by constant takes a single vector of four 32-bit values and an immediate of 5 bits. It performs a shift of each element by the given immediate value. Many block ciphers, stream cipher, and MAC algorithms use add, shift, and xor, and performing each operation in parallel gives a significant speed boost[5].

4.3 Public-Key Acceleration

Public-key cryptography is a useful class of cryptographic algorithms that do not require the initial communication of a secret key. Public-key cryptography is particularly useful for performing key agreement and signature generation, two operations that are common on wireless sensors that need to communicate over a potentially insecure connection.

The most efficient, widely used public-key algorithm, elliptic curve cryptography[14] (ECC), can be extremely slow when implemented in software on resource constrained devices. For example, an optimized implementation of ECDHE takes about 1.15 second of CPU time on the L152RE[27] (a 32MHz Cortex-M3 with no special acceleration for public-key operations). To accelerate the core operations in ECC, CESEL implements a co-processor similar to the one presented in [19].

The core operations for ECC can be identified as the following: modular multiplication and addition. Of these operations, modular multiplication is by far the most time consuming. Montgomery modular multiplication (MMM) is an algorithm that efficiently computes $Mont(x,y) = xyR^{-1} \mod N$. In particular, it avoids the expensive division by N that is required in a naive implementation of modular multiplication. This division is a common bottleneck in other algorithms. MMM also has the advantage of being efficiently implementable in hardware[18].

However, MMM is still quite slow compared to other operations embedded CPUs may perform, typically requiring hundreds of thousands of cycles. Since most embedded processors cannot execute instructions out of order, adding MMM as a primitive instruction would prevent the core from doing any other useful work while the MMM operation is being computed. Thus, to allow ECC operations to occur independently from the main processor, CESEL implements ECC acceleration as a co-processor that can be communicated to from the main processor over the normal memory bus. This allows the CPU to perform other useful work or to sleep during ECC operations. For a complete implementation, the co-processor should be able to compute MMM as well as conversion to and from Montgomery representation and modular addition.

Additionally, ECC signatures are created by first hashing the message. To accelerate signatures, CESEL has a fast hardware implementation of a well analyzed hash function such as SHA-3. Hash functions (especially complicated hash functions, such as SHA-3) can be very expensive to implement on an embedded platform, but quite fast when implemented in hardware[11]. Since signature generation depend directly on the performance of the hash function, a fast and efficient implementation of a hash function will greatly accelerate the verification and generation of ECC signatures.

4.4 Post Quantum Hardware Acceleration

The development of quantum computing present a significant challenge to the long term security of wireless sensors. In particular, Shore's algorithm, a quantum algorithm that enables the efficient integer factorization, violates the fundamental security assumptions behind both ECC and RSA, breaking them entirely[6]. However, not all cryptographic algorithms are made broken by quantum computers; nearly all symmetric key ciphers, hash functions, and MAC constructions remain secure. In particular, the symmetric ciphers and MAC algorithms that CESEL was designed to accelerate (block ciphers based on S-Boxes and vector arithmetic, MAC algorithms based on modular polynomial multiplication and vector arithmetic) are considered secure even in a post-quantum world[6]. Additionally, new post-quantum secure public-key operations have been developed that provide security even in the face of widely available quantum computers.

Unfortunately, most post-quantum algorithms suffer from a number of drawbacks, including limited cryptanalysis, large public and private keys, and slow operation. In our proposal, we focus on one particular construction, *ring learning with errors* (R-LWE)[21][22], which provides a good balance between security, key size, and speed. R-LWE has had extensive cryptanalysis and is considered to be reasonably secure. It is also amenable to hardware implementation and generally free from patents.

To accelerate R-LWE, CESEL has a co-processor similar to the one created in [24]. R-LWE based cryptosystems operate in a polynomial ring $R_q = \mathbb{Z}_q[x]/\langle f(x)\rangle$, where one typically chooses $f(x) = x^n + 1$ with n a power of two, and q a prime with $q \equiv 1 \mod 2n$. Any implementation must implement basic operations in R_q, such as addition and multiplication. In R-LWE, multiplication in R_q is by far the most expensive operation[24].

Therefore, an efficient method for polynomial multiplication will significantly speed up R-LWE cryptographic operations. Implementation in hardware also allows for a number of optimizations that are expensive or impossible to perform in software, such as pipelining intermediate values and fast convolution[24]. However, like ECC, R-LWE is quite slow even when implemented in hardware. Thus, for the same reasons outlined in section 4.3 (i.e. high latency instructions blocking useful work) R-LWE operations must be implemented as a co-processor.

4.5 Approximate Area for Co-Processors

The two largest components in CESEL not found in traditional SoCs are the ECC and post-crypto co-processors. The ECC co-processor in CESEL is based on the implementation in [19], which uses approximately 115,000 gates. The post-quantum co-processor is based on the implementation in [24] which uses approximately 30,000 gates. Thus, the total gate count can be approximated to be

somewhere between 150,000-200,000 gates. As a point of comparison, this is about 0.25mm^2 of die area in a 0.65nm process, roughly 1/16th the amount of area needed for the analog components of Bluetooth[26].

5 Conclusion

The emergence of low power wireless sensor networks presents a large number of security challenges. In this paper we examined the fundamental cryptographic primitives required to secure sensors over a multiple decade lifetime: random number and nonce generation, symmetric key ciphers, and public-key cryptography.

We then proposed a new architecture, CESEL, which accelerates those primitives. CESEL adds five hardware primitives: an atomic, unique counter, a random number generator based on physical entropy, additional microcontroller instructions to accelerate symmetric cryptography, an elliptic curve co-processor, and a co-processor for post-quantum cryptography. This design allows for both long term security and efficiency on a resource constrained device.

6 References

[1] CVE-2011-3389: The SSL protocol encrypts data by using CBC mode with chained initialization vectors, which allows man-in-the-middle attackers to obtain plaintext HTTP headers via a blockwise chosen-boundary attack (BCBA) on an HTTPS session, in conjunction with JavaScript code that uses (1) the HTML5 WebSocket API, (2) the Java URLConnection API, or (3) the Silverlight WebClient API, aka a "BEAST" attack. `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-3389`, September 2011.

[2] CVE-2014-4422: Apple iOS before 8 and Apple TV before 7 uses a predictable random number generator during the early portion of the boot process. `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-4422`, September 2014.

[3] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks: Int J Comput Telecommun Netw*, 38(4):393–422, March 2002.

[4] E. Barker and A. Roginsky. Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. *NIST Special Publication*, 800:131A, 2011.

[5] D. Bernstein and P. Schwabe. Neon crypto. In E. Prouff and P. Schaumont, editors, *Cryptographic Hardware and Embedded Systems CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 320–339. Springer Berlin Heidelberg, 2012.

[6] D. J. Bernstein. Introduction to post-quantum cryptography. In *Post-quantum cryptography*, pages 1–14. Springer, 2009.

[7] A. Bittau, M. Handley, and J. Lackey. The final nail in wep's coffin. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15 pp.–400, May 2006.

[8] A. Cherkaoui, V. Fischer, L. Fesquet, and A. Aubert. A very high speed true random number generator with entropy assessment. In *CHES 2013 15th International Workshop on Cryptographic Hardware*, pages 179–196. TeX Users Group, August 2013.

[9] H. Corrigan-Gibbs and S. Jana. Recommendations for randomness in the operating system or, how to keep evil children out of your pool and other random facts. In *Proceedings of the 15th USENIX conference on Hot Topics in Operating Systems*, pages 25–25. USENIX Association, 2015.

[10] J. B. Daniel and T. Chou. Faster binary-field multiplication and faster binary-field macs. Cryptology ePrint Archive, Report 2014/729, 2014.

[11] K. Gaj, E. Homsirikamol, and M. Rogawski. Fair and comprehensive methodology for comparing hardware performance of fourteen round two sha-3 candidates using fpgas. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 264–278. Springer, 2010.

[12] C. T. Inc. MICAz wireless measurement system. `http://www.xbow.com`, June 2004.

[13] L. R. Knudsen, V. Rijmen, R. L. Rivest, and M. J. Robshaw. On the design and security of rc2. In *Fast Software Encryption*, pages 206–221. Springer, 1998.

[14] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[15] W. Liu, Z. Wang, S. Qu, and R. Luo. Reconfigurable network accelerator for wireless sensor nodes. In *Advanced Communication Technology (ICACT), 2015 17th International Conference on*, pages 138–142. IEEE, 2015.

[16] M. Matsui. Linear cryptanalysis method for des cipher. In *Advances in CryptologyEUROCRYPT93*, pages 386–397. Springer, 1994.

[17] NIST. NIST brief comments on recent cryptanalytic attacks on secure hashing functions and the continued security provided by sha-1. `http://csrc.nist.gov/groups/ST/toolkit/documents/shs/hash_standards_comments.pdf`, August 2004.

[18] S. Ors, L. Batina, B. Preneel, and J. Vandewalle. Hardware implementation of a montgomery modular multiplier in a systolic array. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 8 pp.–, April 2003.

[19] S. Ors, L. Batina, B. Preneel, and J. Vandewalle. Hardware implementation of an elliptic curve processor over gf(p). In *Application-Specific Systems, Architectures, and Processors, 2003. Proceedings. IEEE International Conference on*, pages 433–443, June 2003.

[20] C. Otero. *ASYNCHRONOUS DESIGN FOR UBIQUITOUS COMPUTING*. PhD thesis, Cornell University, August 2014.

[21] O. Regev. New lattice-based cryptographic constructions. *Journal of the ACM*, 51(6):899–942, 2004.

[22] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):34, 2009.

[23] R. Roman, C. Alcaraz, and J. Lopez. A survey of cryptographic primitives and implementations for hardware-constrained sensor network nodes. *Mobile Netw Appl*, 12:231244, October 2007.

[24] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede. Compact ring-lwe cryptoprocessor. In *Proceedings of the 16th International Workshop on Cryptographic Hardware and Embedded Systems — CHES 2014 - Volume 8731*, pages 371–391, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[25] M. Ryan. Bluetooth: With low energy comes low security. In *WOOT*, 2013.

[26] W. W. Si, D. Weber, S. Abdollahi-Alibeik, M. Lee, R. Chang, H. Dogan, H. Gan, Y. Rajavi, S. Luschas, S. Ozgur, et al. A single-chip cmos bluetooth v2. 1 radio soc. *Solid-State Circuits, IEEE Journal of*, 43(12):2896–2904, 2008.

[27] H. Tschofenig and H. Pegourie-Gonnard. Performance of state-of-the-art cryptography on arm-based microprocessors. Presented at the NIST Lightweight Cryptography Workshop 2015, 2015.