UNDERSTANDING AND MANAGING ENTROPY AND RANDOM DATA

January 2016

whitewoodsecurity.com

WHITEWOOD 100 High Street 28th Floor Boston, MA 02110 USA
p. +1.617.391.0268 e. info@whitewoodsecurity.com

UNDERSTANDING AND MANAGING ENTROPY

As Moore's Law marches on, the Internet of Things continues to gain traction, and personal privacy is an above the fold news story, the importance of cryptography on the modern Internet continues to increase. While once the realm of mathematicians and hard core computer scientists, cryptography has gone mainstream. Vulnerabilities such as HEARTBLEED, FREAK and POODLE are reported by mainstream media and cause discussions not just in security teams but dinner tables around the world.

With all this attention on the quality of our protective encryption capabilities, encryption is still a very complicated and not well-understood process. The ecosystem of crypto components that is used in a conventional enterprise is dizzying and even those who try to understand what's going on are stymied by the diversity, age, and code complexity of the various software components.

While cryptographic core algorithms have been well studied, other components in enterprise cryptosystems are less understood. Nearly every crypto system relies heavily on access to high quality random numbers. These random numbers are used for long term key creation, ephemeral key creation, and nonces and IVs that prevent replay attacks and various types of cryptanalysis. If the random numbers used in these crypto systems aren't truly random (or at least random enough to withstand cryptanalytic scrutiny), then the security the algorithm provides can be completely compromised. In an attempt to bring a greater level of understanding to this topic Whitewood Encryption Systems sponsored the KEYW Corporation, a company with deep cryptographic and cryptanalytic expertise, to research entropy production and consumption in common use cases. KEYW has run numerous experiments and audited many open source products in an attempt to find out what's really going on with entropy within the enterprise.

WHAT IS ENTROPY?

Historically, true random numbers have been hard to come by. True random numbers typically come from chaotic physical processes such as thermal noise, atmospheric noise, and even Lava Lamps. While these processes are chaotic and may generate almost truly random numbers, they are relatively low bitrate and can't keep up with the needs of most crypto systems, even assuming they can be accessed at all from inside a locked-down data center.

Provisioning of random numbers in both computers and dedicated crypto systems is typically driven not by quality of randomness, but by a desire to build the cheapest possible circuit that seems random enough. For example, linear congruential generators (LCGs) are used by every major operating system to supply "random" numbers through the built in rand() function. However LCG's have known statistical properties, and are insufficiently random for almost any purpose.

In hardware, chips with a cryptographic core will often use ring oscillators, odd-length of NOT gates whose output feeds back into its input. It can be tricky to guarantee randomness of these circuits, especially as chip feature size decreases, but they are incredibly cheap to build into a chip, and are self-clocking.

To compensate for the lack of high speed true random numbers, system developers turned to Pseudo Random Number Generators (PRNG). These PRNG's conventionally consist of an internal state that updates itself through a fixed rule, and a hash function applied on output to hide the internal state. The data that comes from the hash function looks random and as long as an attacker can't find the internal state of the PRNG and the seeding data is random enough and applied frequently enough, then the data that comes from the PRNG is cryptographically strong enough for use. However, as discussed later, achieving this level of quality with a PRNG is less common that you would hope and expect.

Truly random data has a measurable characteristic called entropy. The term was coined in 1865 to describe a measure of the disorder of a thermodynamic system, and adapted by Shannon in 1948 as a measure of the unpredictability of information content. So right from its inception, entropy was always supposed to be measurable. Unfortunately, there is no single objectively good way to measure the entropy of real data. For example, in 2012 NIST published a suite of statistical tests for entropy sources. In 2015, they reported that those tests had been found to seriously underestimate entropy [3].

The difference between entropy and randomness can be difficult to understand. Probably the easiest way to think about the difference is entropy is the uncertainty of an outcome yet to happen; Randomness is the quality of the uncertainty from a historical perspective. While we can agree on the abstract concept of randomness, the only thing we can quantitatively measure is our ability to distinguish some piece of data from abstract random data, and that depends on our perspective, background knowledge, and how much data we have. In short, it is highly subjective. For a good discussion on entropy in PRNG's and PRNG construction in general, see [4].

Since entropy cannot be fully quantified, our only option is to estimate the entropy of seed data fed in to a PRNG and therefore the quality of its output. In general, the more entropy that is fed in to the PRNG, the more secure the output of the PRNG is presumed to be. If the data that is fed in to the PRNG contains little (or no) entropy, attackers may have the ability to guess the inputs and therefore may be able to divine the output. While low entropy is not a guarantee of compromise, it does weaken the entire crypto system – potentially very seriously. For example, the Debian OpenSSL PRNG debacle of 2008 caused all keys generated and used on a Debian system to be easily breakable[5]. In 2015, an audit of Github SSH keys found that many were still vulnerable [6].

THE ENTROPY LIFECYCLE

Entropy can (and should) be viewed in the context of a lifecycle. In Figure 1, data moves from the land of pure entropy on the left and gets diluted through a series of PRNG's and other activities as entropy is eventually consumed by applications on the right.

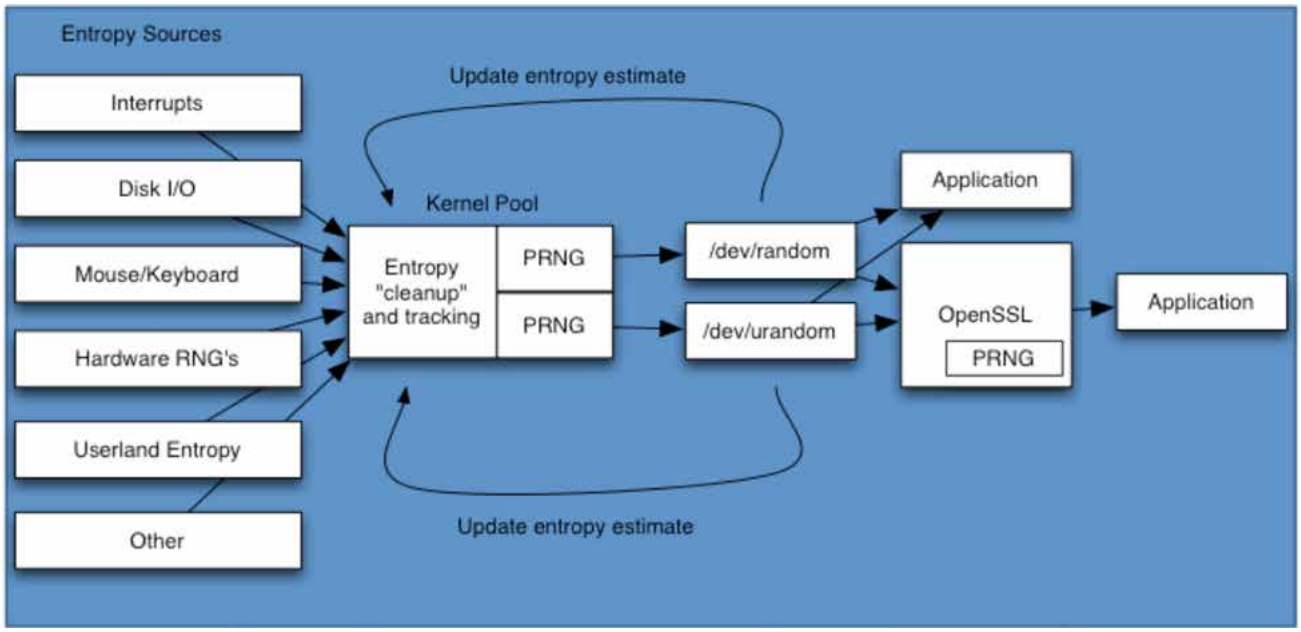

Figure 1 – Entropy Lifecycle

In modern Unix-variants and Linux, there are a numerous sources of entropy available to the kernel. From interrupt timing and network inter-arrival time on the low end, to hardware RNG's such as Intel's Ivy Bridge RNG and Whitewood's Quantum Random Number Generator on the high end, entropy is produced at various rates and qualities. In Linux, this entropy is then fed in to the kernel entropy pool where it is hashed for use via two separate interfaces. /dev/random provides random data that is nearly 100% entropy. If there is no entropy in the pool, /dev/random will block until the OS generates more entropy. /dev/urandom, on the other hand, does not block and will hand out data from its PRNG regardless of the amount of entropy in the entropy pool. The default size of the kernel entropy pool is 4096 bits, but can be modified via a kernel configuration option.

Applications can pull directly from /dev/random or /dev/urandom to get random numbers as needed. One of the largest consumers of random numbers is OpenSSL. OpenSSL provides cryptographic capabilities both through a command line interface and through a library that

is linked in to a wide variety of software packages, including Apache and OpenSSH. OpenSSL pulls data from /dev/urandom only. OpenSSL has its own PRNG that it uses to supply random data for its own cryptographic operations. This PRNG is seeded by a call to /dev/urandom.

ENTROPY PRODUCTION

Entropy production in Linux can come from a variety of sources. Without the assistance of external hardware RNG's, Linux uses naturally occurring chaotic events on the local system to generate entropy for the kernel pool. These events, such as disk IO events, network packet arrival times, interrupt timing, keyboard presses, and mouse movements, are the primary sources of entropy on many systems. The amount of entropy generated from these sources is directly proportional to the amount of activity on each source. It should be noted that for some systems, such as servers and embedded systems, many of these sources of entropy simply don't exist. Most servers do not have keyboards and mice and are therefore unable to gather entropy from these sources. Servers and embedded systems rely on only a small subset of entropy sources, mostly utilizing disk IO events and network packet arrival times.

To explore the production rates of entropy on servers in the datacenter, Whitewood focused on varying the disk IO and network traffic in order to determine the impact of this variance on entropy production. From Whitewood's research, the growth of the pool due to events on the system is relatively slow. Table 1 shows growth rates for various use cases on both virtual machines and bare metal (i.e. non-VM) systems.

Activity	Average Production Rate (Virtual Machine)	Average Production Rate (Bare Metal)
Unloaded system	1.94 bits/sec	2.44 bits/sec
Lightly loaded webserver	3.85 bits/sec	4.84 bits/sec
Heavily loaded webserver	5.56 bits/sec	7.66 bits/sec
Receiving 1000 pings/sec	3.77 bits/sec	13.92 bits/sec
Heavy disk IO	3.26 bits/sec	4.75 bits/sec
RDRAND	Not Applicable	0.58 bits/sec

Table 1 – Entropy Production

While the conventional wisdom generally holds that a bare metal machine generates more entropy than a virtual machine, the difference in generation rate is not exceptional. Only in the case of heavy network traffic does the production rate vary by more than 2x. It should be noted

that this is not a very fast rate of production, particularly if entropy is being drained from the pool at a rate that may even exceed production. Dedicated hardware RNG's on the other hand can fill the entropy pool in <1 second. For comparison, Whitewood's quantum RNG solution can produce entropy at up to 200Mbits/sec.

Of interest in this data is the rate of entropy growth when the Ivy Bridge hardware random number generator (RDRAND) is used. RDRAND can generate random numbers at a rate in excess of 800Mbps. However, in the latest version of the Linux 3.2 kernel, the kernel entropy pool does not increase the entropy estimation for data pulled from RDRAND due to concerns of nation state backdoor possibilities. In fact, from Whitewood's testing, the Linux entropy estimation on an unloaded system using RDRAND is less than on a system not using RDRAND.

Whitewood also examined entropy production on Android devices. The conventional wisdom around entropy on mobile devices is that the devices are poor producers of entropy and represent real concerns for developers attempting to perform cryptographic operations. Whitewood discovered entropy production varied substantially on different devices. Table 2 contains the results of the Android research.

Device and Activity	Average Production Rate
Galaxy S3, IO Load	20.7 bits/sec
Galaxy S3, IO Load and web browsing	24.4 bits/sec
HTC one M8, IO Load	38.4 bits/sec
HTC one M8, unloaded	23.7 bits/sec
Apex (atom based tablet), unloaded	23.7 bits/sec
Proprietary Phone, unloaded	38.5 bits/sec

Table 2 – Entropy Production on Android Devices

The research challenges conventional wisdom. Every Android device we tested generated an order of magnitude more entropy than their server counterparts. While the entropy production rate still doesn't approach that of hardware RNG's, Android devices fill their entropy pools much faster than modern servers.

ENTROPY CONSUMPTION

Entropy consumption can vary wildly depending on the application and specifics of the cryptographic operation. For this research, we examined common use cases involving OpenSSL including Apache and nginx webservers as well as common OpenSSL crypto operations such as key generation.

OpenSSL Concerns

When it is initialized, OpenSSL attempts to pull 32 bytes of random data from the non-blocking interface, /dev/urandom. Whitewood audited OpenSSL and determined OpenSSL does not verify how much entropy is in the pool and assumes the data is acceptable. From our research, if there are less than 384 bits of entropy in the kernel pool, /dev/urandom will not return data with 32 bytes of entropy. In fact, if the kernel pool has less than 192 bits of entropy, the kernel entropy estimation isn't decremented at all. This means that depending on the state of the kernel entropy pool when OpenSSL is started, little to no new entropy will be contained in the random numbers provided from the /dev/urandom PRNG.

OpenSSL only seeds its internal PRNG once per runtime. This is not an issue for short lived executions, such as command line invocations to create a single key. However, for long running processes, including the common scenario when web servers link to the OpenSSL libraries, it means that all PRNG operations performed for the duration of the server only have as much entropy as was available at the invocation of the OpenSSL library.

Mod_SSL Concerns

Mod_SSL is the primary mechanism for servicing TLS connections in the Apache webserver. Mod_SSL is effectively a library that interfaces directly to OpenSSL to provide all SSL and TLS functionality to Apache. Mod_SSL attempts to add its own entropy to OpenSSL to address the lack of reseeding that occurs within OpenSSL. For every request, Mod_SSL attempts to add entropy by concatenating the Program ID (PID), time, and 256 bytes from the stack and then stirring that value in to the internal OpenSSL PRNG.

This operation adds very little entropy. The PID for webservers is long running and does not change over time, and the date value used in this calculation only has a resolution of one second so it changes very slowly (and regularly). The value from the stack could in principle be a source of large amounts of entropy, but that does not seem to be the case. Based on Whitewood's analysis, this stack value only changes ~8% of the time it is queried. All other times, the value from the query is the same as the previous value. While 256 bytes is clearly a lot to guess, the goal of adding entropy to the PRNG is usually not successful.

Nginx Concerns

While Mod_SSL at least attempts to add entropy to OpenSSL's PRNG, nginx does not. By default it never reseeds the PRNG with any entropy at all. It can be configured to add entropy directly through OpenSSL by calling OpenSSL's reseed function. However since this is not the default behavior, it is unclear how often administrators chose to perform this reseed.

ENTROPY AND RANDOM NUMBERS

Applications don't use entropy directly, they use random numbers. A PRNG, once seeded with entropy, can be used to return a vary large amount of random numbers for use by an application. How many random bytes are generated is not well understood and the correlation to bits of entropy even less so. Given our findings with Apache, OpenSSL, and nginx, it seems that the correlation of bits of entropy to discrete operations is vague at best. Whitewood instrumented OpenSSL in an attempt to at least determine at a transactional level how many random bytes are required to perform standard cryptographic operations. Figure 3 shows the averages of our research as performed on a server platform.

Activity type	Average Number of Random Bytes Required
RSA 1024 bit key gen	5721 bytes
RSA 2048 bit key gen	14696 bytes
RSA 4096 bit key gen	139315 bytes
Apache TLS with 1024bit RSA key	498 bytes
Apache TLS with 1024bit RSA key and Perfect Forward Secrecy	635 bytes

Table 3 – Random Number Requirements

With key generation, our research indicates dramatic increases in random number requirements as key size increases. Generating a 4096 bit RSA key requires almost 25x the amount of random data as generating a 1024 bit key. And while individual Apache requests seem to not have excessive random number requirements, keep in mind that Apache does not have a default for maximum number of connections a server can process. That means a moderately loaded webserver with 100,000 connections/day could require nearly 2GB of random data in a month. Without modification, Apache will continue to use the initial seed data forever and will base all require random numbers off that seed.

UNCERTAIN INPUTS MAKE UNCERTAIN RESULTS

Whitewood's research uncovered many concerns with how modern systems handle entropy and random number generation. In fact, conventional wisdom has been challenged when it comes to virtual machine versus bare metal entropy production as well as the reseeding capability of major webservers.

Cryptographic systems rely on correctness in order to protect our data. Minor variances from standards and best practices can lead to catastrophic failure. While none of Whitewood's findings are a single smoking gun that lead directly to data compromise, there is a common theme of misunderstanding and mismanaging entropy in modern software systems. The failures in entropy management fall in to three main areas:

Quality – From our research, we see variation in the quality of entropy. In an attempt to reseed the OpenSSL PRNG, Apache attempts to add entropy from several sources. Unfortunately these sources are relatively static and predictable and do little to produce quality data for reseeding. On the flip side, we see the Linux kernel assuming NO entropy due to input from RDRAND even though Intel and others have extensively tested the implementation and found it to be a source of high quality entropy for use in cryptographic systems.

Quantity – Without hardware RNG's, most modern systems produce entropy on the order of bits per second. For cryptographically intense systems such as webservers and VPN routers, the volume of cryptographic operations ensures that the kernel entropy pool is always under heavy pressure and constantly in need of more entropy.

Consistency – Different systems produce and consume entropy at different rates. There is currently no mechanism for monitoring entropy on a consistent basis across multiple systems, and ensuring that all cryptographic operations have the same level of high quality and high quantity entropy is not possible today. Enterprises run on faith that the random numbers needed to perform cryptographic operations are "good enough"

WHITEWOOD'S SOLUTION

Whitewood has focused on addressing the issue of entropy management which the findings presented in this paper have highlighted – in particular the issues of quality, quantity and consistency of entropy generation and distribution to cryptographic applications. At the core of these solutions is the Whitewood Entropy Engine™ a quantum powered random number generator based on technology originally developed at the Los Alamos National Laboratory.

WHITEWOOD ENTROPY ENGINE ™

The Whitewood Entropy Engine solves the problem of entropy generation. It provides provably random data in a convenient PCIe card form factor. At its core is a patent pending quantum entropy source that exploits the immutable laws of quantum mechanics to create truly unpredictable entropy. Capable of delivering 200Mbps the Entropy Engine can satisfy the demands of even the highest performance cryptosystems.

The Whitewood Entropy Engine

PLATFORM FOR ENTROPY-AS-A-SERVICE

Even with high quality entropy, enterprises need a scalable way to deliver entropy to applications that require it. While hardware RNGs can be dedicated to individual servers for critical applications the issue of entropy management is a universal one. Delivering entropy in a scalable and consistent way will often require that a centralized service infrastructure is established. Whitewood addresses this need with a new service platform that enables customers to establish their own entropy-as-a-service (EaaS) capability to ensure consistency, scale and flexibility as they address their entropy needs.

The EaaS platform which incorporates the Whitewood Entropy Engine can be used to supply random data for direct consumption by network connected devices or applications or can used instead to seed remote PRNGs that in turn are used to supply their own local applications.

The Whitewood Entropy-as-a-Service Platform has the following core capabilities:

- Support for the Whitewood Entropy Engine – best in class quantum powered random number generation
- Secure network delivery – authenticated, integrity validated and confidential random data prevents eavesdropping and data manipulation
- Central dashboard – shows overall entropy consumption
- Whitewood Entropy Client – provides easy selection of remote entropy sources and consumption statistics
- Open source client code – enables scrutiny of security properties and easy application integration
- Simplified client enrollment – rapid deployment and scalability for individual clients or groups

ENTROPY MANAGEMENT FOR OPENSSL

This paper has highlighted the universal issue of entropy management and in particular the challenges associated with OpenSSL. To address this Whitewood has released a solution to specifically target OpenSSL. The Whitewood Entropy Management for OpenSSL Client is available as a free open-source library that requires no modification to OpenSSL other than an updated configuration file. The library can be used as a standalone agent to get a clearer understanding of the local entropy situation and includes full support for Whitewood's Entropy Engine and can act as a client to the Whitewood Entropy-as-a-Service Platform to access high quality entropy over a network - both of which dramatically improve the supply and quality of entropy for OpenSSL deployments.

By deploying the Entropy Management for OpenSSL Client the user can:

- Select which source of entropy to use for OpenSSL –
  - /dev/urandom
  - RDRAND
  - Whitewood Entropy Engine QRNG
  - Whitewood Entropy-as-a-Service Platform
- Select whether to use the entropy sources directly as a random data stream or as a seed to a Whitewood provided PRNG
- Select between two NIST specified (SP800-90A) PRNGs (CTR DRBG and Hash DRBG)
- Select PRNG configuration (hash functions, AES ciphers, security strength) and operational parameters such as reseeding rate
- Combine PRNG random output with raw entropy to further increased security and resilience
- Monitor local entropy consumption and generation and access statistics via command line
- Easily enroll individual or groups of OpenSSL instances with the Whitewood centralized entropy management server

The Whitewood library for OpenSSL which is known as the WES-entropy-client, has two major components as shown in Figure 2. The first part is the libWES library itself, which acts as a replacement for the OpenSSL random number engine and PRNG. When installed and configured, it handles all requests for random numbers made by OpenSSL.

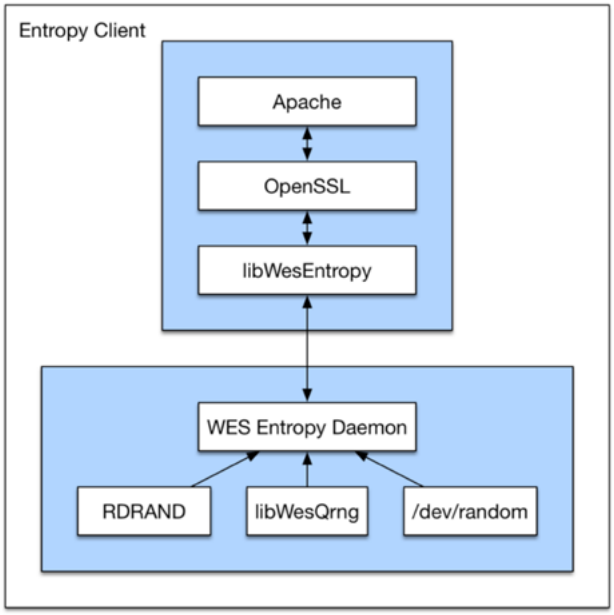

Figure 2 – Whitewood Entropy Management Client for OpenSSL

The second component is the WES entropy daemon. The daemon can be configured to pull entropy from a variety of entropy sources including traditional kernel sources such as dev/urandom, local hardware based RNG's such as RDAND and dedicated RNG's such as the Whitewood Entropy Engine QRNG. The daemon continuously feeds entropy into the libWES library to reseed the Whitewood PRNG. This system addresses the reseed concerns with OpenSSL and allows Apache and nginx to be operated securely for long durations.

The Whitewood Entropy Management Client for OpenSSL has been released as an open-source component and is available for download from https://github.com/WhitewoodCrypto/WES-entropy-client or from www.whitewoodencryption.com.

CONCLUSION

Random number generation has always been a dark art, as has the practice of designing robust entropy processes to capture and assess entropy. And yet, these same random numbers underwrite the security of every modern communications system, including the Internet.

Crypto designers and security professionals alike are expected to know about cryptographic algorithms and key sizes, and their impact on security. But most are still not fully aware of how keys are generated for those algorithms, or how entropy is accumulated, even though the underlying algorithms and technology use well-known software and hardware components.

This paper highlights some truly surprising discrepancies between what people think is happening in commonly used PRNGs, and what is actually happening. It is our hope that this research will encourage others to take a look at their PRNGs, and finally bring this subject into the light where it belongs.

References

[1] – FREAK - https://www.us-cert.gov/ncas/current-activity/2015/03/06/FREAK-SSLTLS-Vulnerability
[2] – POODLE https://www.us-cert.gov/ncas/alerts/TA14-290A
[3] – How Random is Your RNG https://archive.org/details/How_Random_Is_Your_RNG_SC2015
[4] – Pseudorandom Number Generation, Entropy Harvesting, and Provable Security in Linux http://www.blackhat.com/presentations/bh-europe-04/bh-eu-04-hardy/bh-eu-04-hardy.pdf
[5] – http://rdist.root.org/2009/05/17/the-debian-pgp-disaster-that-almost-was
[6] – https://threatpost.com/audit-of-github-ssh-keys-finds-many-still-vulnerable-to-old-debian-bug/113117

JANE E. (BETH) NORDHOLT is a retired Fellow of the Los Alamos National Laboratory and Senior Adviser to Whitewood Encryption Systems, Inc. After Beth and Richard Hughes invented of the methodologies that make long-distance, free-space quantum cryptography (QC) possible, she became co-lead of the LANL QC team and, with Richard, began to develop its free-space and fiber-optic QC capabilities. She is also the inventor of satellite-based QC, which is being integrated with satellite optical communications and pursued in many countries.

While at LANL, Beth was an inventor on 31 patents, patent disclosures and foreign patent applications related to QC and a patent on the mass spectrometer she developed for the NASA-ESA Cassini mission to Saturn. With U.S. Government personnel, she wrote two reports on the security of QC. Beth earned six LANL Distinguished Performance Awards, five of which were related to QC, and also received four LANL Awards Program prizes. Beth started the effort to increase fiber QC distances and security by using specialized detectors in collaboration with personnel from NIST Boulder. This effort resulted in several world-record distances for QC in optical fiber. She was the lead inventor of the Velocirandor quantum random number generator, now called the Entropy Engine, and QKarD, a device and architecture designed to make QC commercially-viable.

Before concentrating her research efforts at LANL on QC, Beth was principal investigator on the NASA Genesis Concentrator, the NASA Deep Space 1 (DS1) Plasma Experiment for Planetary Exploration (PEPE), and led the NASA-ESA Cassini Ion Mass Spectrometer invention and development as part of the CAPS instrument. She received NASA Group Achievement Awards for her work on the Polar/TIDE, Cassini/CAPS, and DS1/PEPE spacecraft, as well as the DS1 flyby of comet Borrelly. She has been a member or chair of several NASA proposal review panels and was co-Lead on the NASA New Millennium Project's Instrumentation team.

RICHARD HUGHES, PhD is a Consulting Physicist and a Senior Advisor to Whitewood Encryption Systems, Inc. of Boston, MA. Richard retired from a three-decade career at Los Alamos National Laboratory (LANL) in 2014, where he held the position of Laboratory Fellow in the Physics Division. Richard founded and for two decades was co-lead of the Quantum Communications team at LANL. He was co-principal investigator of multiple research projects until his retirement. Richard received his B.Sc. (Hons., Class I) degree in Mathematical Physics from the University of Liverpool, England, and his Ph.D. in Theoretical Elementary Particle Physics from the same institution. He held research positions at: Oxford University and The Queen's College, Oxford, England; the California Institute of Technology, Pasadena, California; CERN, Geneva, Switzerland; and the University of Edinburgh, Scotland; before joining LANL as a Technical Staff Member. Richard has held distinguished visiting scientist positions at the University of Oslo, Norway, and at Oxford University (Dr. Lee Fellow, Christ Church). In 1996, 1998, 2006, 2010 and 2012 he was awarded Los Alamos Distinguished Performance Awards for his quantum cryptography research, and in 1997 he was awarded the Los Alamos Fellows' Prize for his research on quantum information science. Richard is a Fellow of the American Physical Society. In 2001 he was co-winner of an R&D100 Award for Free-space quantum cryptography. Starting in 2001, Richard led the US Government's Quantum Information Science and Technology Roadmap. In 2004 Richard and the LANL Quantum Communications Team were co-winners of the European Union's Descartes Prize. He has acted in an advisory role on multiple occasions for several US Government agencies, and in 2008 he received the ODNI Distinguished Analysis Award. Richard has given many invited presentations at major international scientific conferences and research universities. He has 31 US and foreign patents and patent applications in quantum communications, and he has authored over 160 scientific papers on elementary particle physics, quantum field theory, the foundations of quantum mechanics, quantum cryptography and quantum computation.

WHITEWOOD 100 High Street 28th Floor Boston, MA 02110 USA
p. +1.617.391.0268 e. info@whitewoodsecurity.com

ABOUT WHITEWOOD ENCRYPTION SYSTEMS®

Whitewood® is addressing one of the most fundamental challenges associated with all modern cryptosystems – entropy management. Whitewood's products exploit quantum mechanics to meet demand for high-quality entropy used for random data and key generation at scale. Building upon a base of quantum cryptography capabilities developed over the course of the past two decades at Los Alamos National Laboratory, Whitewood addresses operational vulnerabilities in any application that employs encryption, certificates and keys in clouds, devices and browsers. Whitewood is part of Allied Minds Federal Innovations, the division of Allied Minds dedicated to commercializing U.S. federal intellectual property. More information on Whitewood can be found at www.whitewoodencryption.com.

whitewoodsecurity.com

WHITEWOOD 100 High Street 28th Floor Boston, MA 02110 USA
p. +1.617.391.0268 e. info@whitewoodsecurity.com