CRYPTOGRAPHICALLY SECURE TRANSACTIONS WITH OPTICAL CARDS

Jack Harper, President[1]
BSI2000, Inc.[2]
12600 West Colfax Avenue, Suite B410
Lakewood, Colorado 80215 USA
(jharper@bsi2000.com)

Presented at CardTech/SecurTech 2003
Orlando, Florida USA
May 14, 2003

ABSTRACT

A protocol is defined that enables optical cards to be used in a cryptographically secure manner for the first time. The protocol, which involves special crypto hardware, allows optical card systems to have data security that is superior to that provided by smart cards. In addition, the protocol makes it possible to build highly secure card systems without having to implement an expensive and inherently on-line Public Key Infrastructure (PKI). The protocol is designed to work with new optical card systems and to also bring the security of older legacy card systems up to the Federal FIPS 140-1(1,2,3) level.

OPTICAL CARDS – WHAT ARE THEY?

Optical Cards are the same size and shape as standard plastic credit cards, but hold up to four-megabytes -- *about 1,500 pages of typewritten information* -- of updateable digital data in a secure, inexpensive, and compact personal package.

Optical cards hold on the order of 1,000x the amount of information as the typical *smart card* and the data, once written, is permanent and cannot be erased or altered in any way.

Optical cards, unlike smart cards, are also impervious to electric and magnetic fields and also to static electricity.

---

[1] *Messrs. David Burlingame, Glenn Junik, Jay Miller, Ron Miller,* and *Kevin Wilson, Ph.D.,* all of BSI2000, Inc., read the preprint of the paper. They had useful comments and suggestions and found some errors. Any errors that remain are those of the author. *Mr. Jay Miller* of BSI2000, Inc. gathered the empirical data, performed the analysis, and wrote Appendix I on the hardware random number generator.

[2] BSI2000, Inc. www.bsi2000.com (a public company – OTC *BSIO).*

They are not damaged by the usual bending or flexing and have a ten-year life with normal use[3].

The large amount of reliable and permanent memory means that optical cards can easily hold information such as high-resolution digital photos of the cardholder (or of each of the cardholder's family), multiple biometrics (e.g., all ten fingerprints, both irises, both hand geometries, etc.), medical information, citizen information, secure site access rights, and many others[4].

For example, fingerprint biometrics for all ten fingers, iris biometrics for both eyes, and a high-resolution color photograph of the cardholder would use far less than 1% of the capacity of an optical card.[5]

However, perhaps the most important item that can be written to an optical card is a *permanent detailed audit trail* of *every* transaction that involves the card.

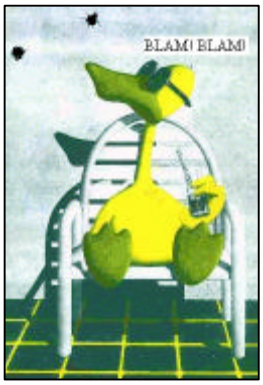
Optical cards are extensively used today, as clearly proven technology, in real projects such as the *INS Green* and *Border Crossing* cards, the *Italian National ID* card project, the *Saudi Arabian National ID* card (coming soon), the *Canadian Permanent Resident Card* ("Maple Leaf") project, and others. Approximately twenty million optical cards are expected to be in use by the end of 2004 in North America alone.

SECURITY WITH OPTICAL CARDS.

Optical cards are wonderful things.

All of the information needed to enable secure transaction systems to function can be carried in a simple way on optical cards rather than in the large, complex, and *expensive* on-line systems that are inherent with smart cards.

However, optical cards have been criticized in the past when used in highly secure applications, as there has not been – *until now* – any practical way to securely protect *secret keys* with optical card-based systems.



Modern cryptographic methods depend on the exchange of *keys*, usually 128 or 2,048 bits in length, to protect data. The larger keys actually come in pairs – a so-called *public key* that is freely released to anyone, and a *private key* that is closely protected and not released.

[3] The United States Department of Defense completed an extensive series of durability and ruggedness tests (*electromagnetic, electrostatic discharge, altitude, extreme temperatures, vibration, water immersion, humidity, icing, salt fog, dust and sand abrasion, solvents, oils, multiple insertions*, etc.) at *Fort Huachuca, Arizona* in September 1998. Optical cards passed these comprehensive tests with a *perfect score*. See http://www.bsi2000.com/downloads.htm for a copy of the report.

[4] ICAO and NIST recommend that original images of the cardholder's face, fingerprint(s), and irises be stored. Biometric templates must be stored as well.

[5] Storing cardholder fingerprints in giant government databases is *politically problematic.* Optical cards essentially eliminate the issue because prints can be stored *only* on the card that is carried by the user.

The private key must be saved in a secure location preferably in specially designed cryptographic hardware.

One characteristic of a good crypto system is that the electrical schematic diagrams, computer software source code, and all other technical details can literally be published in the *New York Times* and the security will not be compromised. Everything can be published – except the *secret key* on which the overall security of the system depends. The secret key must be securely saved in a hardware *tough nut* somewhere within the system.

Furthermore, in a well-designed crypto system, the security of the entire system will not be compromised with the loss of a single key. There should be layers of security – like the layers of an onion – that must be breached, one at a time, in order to penetrate to the protected information.

An interesting issue with building secure crypto systems with optical cards is: *Where do you store the secret key??*

The problem is that data on a basic optical card can be fairly easily read. Such data bits on the card are about two microns in diameter – about one-third the size of a red blood cell – and they are recoverable.

Several methods have been used in the past to try to overcome the secret key problem; each of them is badly flawed:

(a) *Embed the secret key in the software.* However, with this method, an attacker can easily reverse engineer the software object file to recover the key. This method compounds the security problem, as megabytes of software must be protected, rather than just the 128-bit *symmetric*[6] or the *2,048-bit private key.*

(b) *Obfuscate the key by embedding it (or its function) in the 'normal' electronics* – e.g., somewhere in the microcode of the hardware. However, an attacker can reverse engineer the electronics and control software/microcode to recover the key or the crypto function. While not quite as easy a task as reverse engineering pure software, it is still quite doable. This method, however, further compounds the security problem as the hardware and its microcode must be protected from theft.

[6] *Symmetric encryption*, also called *conventional encryption*, is the general method of encryption that uses the same key to both encrypt and decrypt information. *Public key encryption* is the general method that uses two (typically) different keys to encrypt and decrypt – a *private key* and a *public key.* Each general method has advantages over the other – symmetric encryption is much faster but public key encryption eliminates having to, somehow, securely transmit keys to the receiving party – a major advantage.

(c) *Use a crypto method that does not use keys* – something *home grown* perhaps. This method is inherently insecure as there is no effective way to guarantee that a homegrown algorithm is secure without thorough peer-review and deep analysis over many years by qualified cryptanalysts. An additional insidious side effect is that if the keyless algorithm is compromised, then the attacker will achieve complete penetration of the system with potentially disastrous results.

(d) *Embed a smart card chip into the optical card to result in a hybrid card.* This method more than doubles the cost of the card system. Moreover, it loses the simplicity of a standalone system by requiring that the system be inherently on-line. An additional factor is that many smart card chips are really not all that secure against a well funded deep attack[7].

(e) *Some combination of the above.* The *obfuscating kaleidoscope solution* – the worst of all possible worlds. Good crypto systems are conceptually simple – simple enough to be thoroughly understood. Complexity may yield a *feeling* of security but the reality is that almost always, quite the opposite is true[8].

All of this brings up an interesting question – *How much security is necessary?* The answer is simple – *As much as you can afford!*[9]

[7] Smart card chips frequently employ some form of flash memory that can be read simply by shaving the outer housing and then illuminating the die with a scanning electron microscope to read the bits – e.g., see the March 2002 issue of *Information Security Magazine.* Also, see *Tamper Resistance – a Cautionary Note* by Anderson and Kuhn (http://www.cl.cam.ac.uk/~mgk25/tamper.html) for an excellent – and scary – introduction to the problem.

[8] Some people believe that *MicroSoft Windows*? is a good example of this phenomenon. Something, perhaps, is amiss when a basic instruction manual to use the system is longer than the actual source code of earlier PC operating systems.

[9] This implies, of course, that as better security becomes available for the same cost, then you should upgrade legacy systems to keep up with the ever-increasing threat.

THE *SECURE OPTICAL CARD PROTOCOL - SOCP*[10]

BSI2000 has, over a period of some years, developed its own unique solution to the problem of building *cryptographically secure* systems that use optical cards.

The first step was to identify the required characteristics of a cryptographic protocol that is designed to enable secure transactions with optical cards. The design parameters were as follows:[11]

(a) The embodiment of the protocol must provide protection for the data stored on optical cards to the Federal *FIPS 140-1 (1, 2, 3)* level. In short, it must enable optical card systems to operate in a credible *cryptographically secure* manner[12].

(b) The invented cryptographic protocol must enable standard public key cryptography to be used to provide security for data that is written to optical cards that has *even better protection* than that provided by the highest level (i.e., most expensive) standard smart card chips.

(c) The protocol must allow data that is written to any card *(1, 2, …, m)* to be securely read only by any of the *optical card transaction processing TPU units (1, 2, …, n)* in the distributed network (see photo at the top of this section for an example TPU that is built by BSI2000, Inc).

(d) The protocol cannot depend on *obscurity*[13] and, assuming a perfect technical implementation, the protection provided to the overall system must depend only on the level of protection and security of the *secret keys.*

(e) The protocol must operate securely over a network of *n* optical card TPUs where the units are interconnected either solely by data being transferred on optical cards, or perhaps, also by standard local area networking or by Internet networking techniques, or perhaps, by all of the above.

[10] Patent Pending.

[11] While a good crypto protocol is, among other things, simple enough to completely understand, it is interesting that the design requirements of a good protocol may *not* be quite so simple.

[12] Previous optical card systems have not been able to operate in a cryptographically secure manner.

[13] A naïve approach to security is to scatter a key around the inside of a software system – such a method does *not* provide true security at all. The key can, and will, be recovered simply by reverse engineering the system usually within only a few weeks or months of concerted effort..

(f) The protocol must be able to *authenticate* data meaning that a TPU receiver of a data record must be able to securely determine the true origin of the record.[14]

(g) The protocol must provide for *data integrity* in that it must be possible for a receiving TPU to securely verify that a data record has not been surreptitiously modified in transit on an optical card or network.

(h) The protocol must provide for *no repudiation* in that it must not be possible for a TPU to falsely deny having written a data record to an optical card.

(i) The protocol must minimize the possibility of a *known plaintext*[15] or similar cryptoanalytical attack.

(j) The protocol must not require an *arbitrator* or similar on-line device but, rather, must function securely completely *off-line* where the only communication between TPU transaction units is by optical cards. In particular, the protocol must not require the presence of a *Public Key Infrastructure* (PKI) system in order to operate securely. Such PKI systems are difficult to build and are quite expensive to operate over time.

(k) The protocol must be immune from *man in the middle* attacks.

(l) The protocol must be resistant to so called *rubber hose cryptoanalysis* where attempts are made to coerce secret keys from holders. Losing a single key or physical component should not compromise the overall system.

(m) The protocol must guard against illicit duplication and use of optical cards (i.e., *clones)* and against illicit cloning of data records that are duplicated either from within one card or from a group of multiple cards.

[14] "Authentication is more important than encryption. Most people's security intuition says exactly the opposite, but it's true. Imagine a situation where Alice and Bob are using a secure communications channel to exchange data. Consider how much damage an eavesdropper could do if she could read all the traffic. Then think about how much damage Eve could do if she could modify the data being exchanged." -- from *Bruce Schneier's Crypto Newsletter, February 2003*. See *http://www.counterpane.com/crypto-gram.html.*

[15] *Known Plaintext, Man in the Middle,* etc. are the names of well-known crypto attacks. If you know the data that is encrypted (e.g., a name of a particular person), then to recover the crypto key is far easier.

(n) The protocol must minimize the probability of loss of overall security of the TPU network if one or more individual TPU devices are stolen and reverse engineered.

(o) The technical embodiment of the protocol should be as resistant to reverse engineering as practical by using *Tough Nut* hardware protection.

(p) The underlying special support hardware – the *Tough Nut* that protects the keys – in the embodiment of the protocol must be affordable for *all* applications.

(q) The protocol must be efficient with no requirements for either limited or massive numbers of pregenerated data objects (certain *digital cash* protocols).

(r) At least three separate pieces of hardware and a text pass phrase must be stolen or compromised in order to have any realistic chance at all of beginning to defeat the protocol.

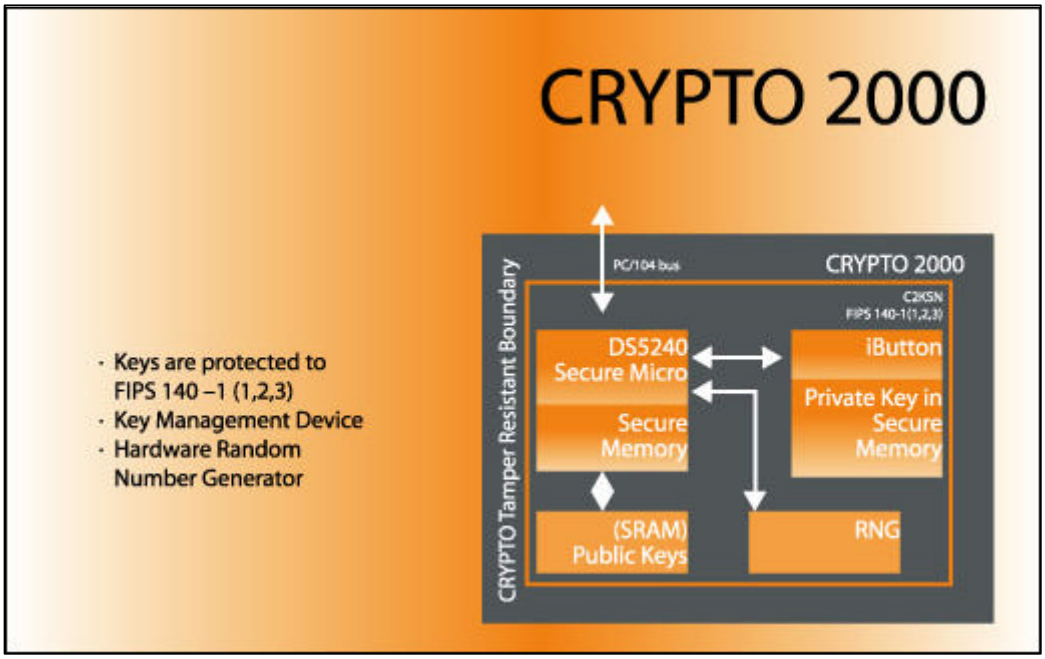
HARDWARE STRUCTURE.

The foundational element of the secure optical card protocol in its current embodiment is the *Crypto 2000* hardware board that has been recently announced by BSI2000, Inc.[16] The board has the *PC/104* form factor[17] and is designed to simply plug into any of the BSI2000 family of optical card TPU units as an inexpensive high security option.

The *Crypto 2000* has three primary functions that are integral to the cryptographic protocol:

(a) To serve as a *secure*[18] repository for secret keys.

(b) To serve as a *secure key management device* that includes generation and encryption - decryption of keys and key pairs.



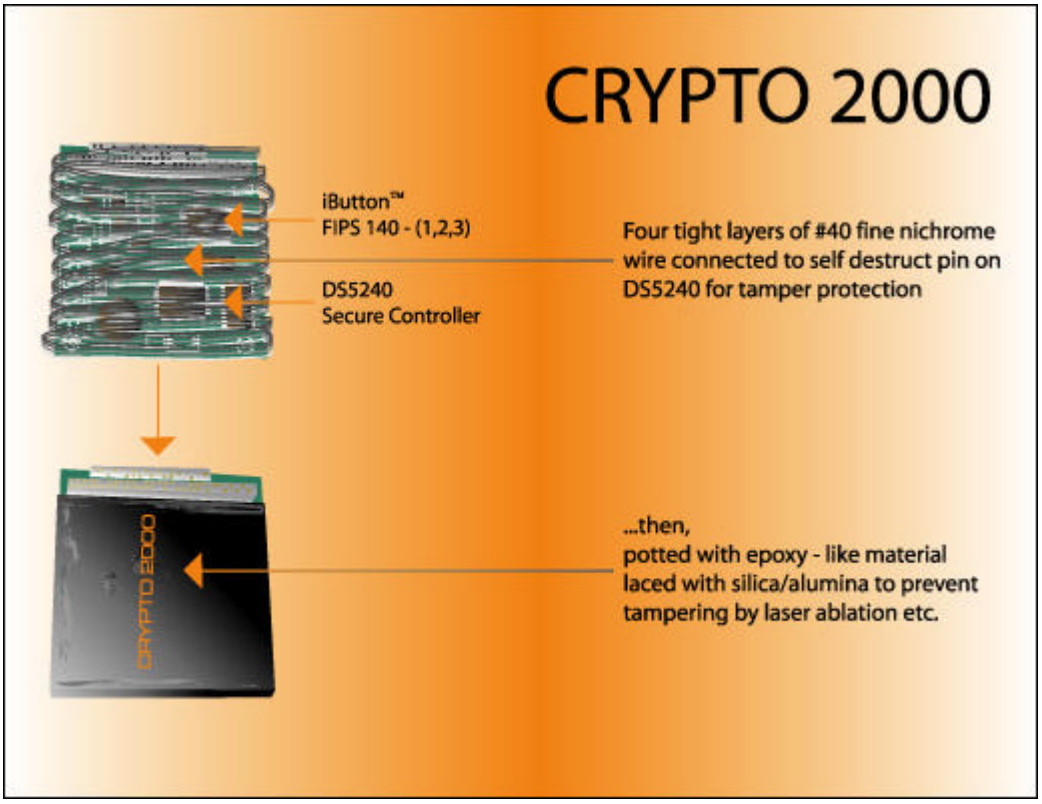
[16] Patents Pending.

[17] The PC/104 standard is commonly used in the embedded systems market. The boards are small (3.6x3.8 inches), rugged with gas-tight connectors and other features for use in difficult and harsh industrial environments, and are stackable with 120 interconnecting pins.

[18] The underlying *Tough Nut* is *FIPS 140-1* (levels 1, 2, and 3) compliant.

(c) To serve as a demonstrable source of *cryptographically secure random numbers* (prime and otherwise) for keys and key pairs that are generated by clearly understood and carefully monitored physical effects that are driven by underlying quantum physics.

KEY MANAGEMENT HARDWARE –
THE TOUGH NUT…



The Crypto 2000 board includes the *FIPS 140-1 (1, 2, 3)* certified *tough nut*[19]device – the *iButton*™ available from *Dallas Semiconductor*[20] – that reposits the secret key for the particular unit.

The Crypto 2000 also includes the *DS5240 Secure Microcontroller* chip (also from Dallas) that securely protects the decrypting public keys for all of the other Crypto 2000 units in a network.

The pair of secure devices operate in concert to enable networks of thousands of TPUs with millions of secure optical cards to operate in a *cryptographically secure manner.*

[19] You have to be very careful even when using *Tough Nuts.* The paper referenced in an earlier footnote – *Tamper Resistance – a Cautionary Note* – describes a successful attack on the older *DS5002FP Secure Microcontroller.* The attack depended on side effects of encrypted instructions and involved, initially, the construction of a codebook of encrypted instruction byte groups. Eventually, the attacker developed the meaning of enough encrypted instructions to enable construction of a small program that dumped the protected data to a logic analyzer. It is an interesting approach and clearly illustrates that more time should be expended trying to break a system than was expended in designing and building it. Crypto 2000 certainly takes this attack into account.

[20] http://www.maxim-ic.com

The *iButton* and the *DS5240* are specifically designed to provide an on-chip self-contained cryptographic boundary that is tamper reactive and able to securely store and manage secret keys within the special hardware. The devices are designed to resist multiple levels of threat including observation, analysis, and physical attack.

The *iButton* and the *DS5240* combine to provide three fundamental security elements:

(a) *Fast and Complete Zeroization:* An attacker's first target in an embedded crypto system is usually the physical memory[21]. Some embedded systems and smart cards attempt to achieve at least some security by using microcontrollers that have internal floating-gate memory (EPROM or FLASH). To erase floating-gate memory cells requires considerable time for both UV-erasable EPROM and FLASH memory which is certainly not acceptable for secure applications.

Even worse – *in fact, much worse* – is the fact that floating-gate technologies are intrinsically nonvolatile and maintain their cell contents when power is removed from the microcontroller.

The decay time for the data is measured in hundreds of years which certainly gives attackers time to breach the physical defenses in the chip[22] to access the protected information even when not powered up.

If the physical package of the iButton or the DS5240 chip is penetrated by any known means, then *complete* zeroization of the keys occurs almost instantly. The same zeroization occurs when any of the protective on-chip and/or off-chip tamper detection systems, next described, are activated.

(b) *Tamper Sensors:* The devices incorporate a number of *tamper resistant features* to protect against physical attack. The primary tamper resistant feature of the chips in each device is an additional metal layer die top coating that is designed to prevent *microprobe attacks* on the chip itself even when the chip is not powered.

[21] A simple logic analyzer can easily monitor and decode all data moving on address and data buses.

[22] Contents of many smart card chips are "routinely" read simply by shaving or ablating off the surrounding structure and then reading the naked cells with a scanning electron microscope that can easily be rented. The author saw one for sale recently on *eBay*.

The layer is *"a complex layout that is interwoven with power and ground which are in turn connected to logic for the Encryption Key and Security Logic. As a result, any attempt to remove the layer or probe through it will result in the erasure of the security lock and/or the loss of encryption key bits"* (from Dallas documentation).

The tamper response, when activated, instantly erases the internal encryption keys, interrupt vector tables, and all data that is stored in the battery backed up internal code/data SRAM memory. In addition, two general purpose *self destruct input pins* on the DS5240 chip automatically cause almost instantaneous and complete zeroization of the protected secure memory when their lines are disturbed even when the unit is not powered. One of the pins is connected to external off-chip *tamper sensors* that are configured inside the BSI2000 TPU transaction processing unit case.

The other input pin is connected to a four layer wrapping of brittle 40-gauge wire that surrounds the entire Crypto 2000 board which is, in turn, embedded in a block of hard opaque epoxy-like material that is mixed with ground silica, alumina, or a *filled encapsulate* that makes it extremely difficult to machine or laser ablate the surrounding block.

The active tamper sensors make it extremely difficult to physically get anywhere near the IC package without triggering the automatic zeroization mechanism that obliterates the secret keys.[23]

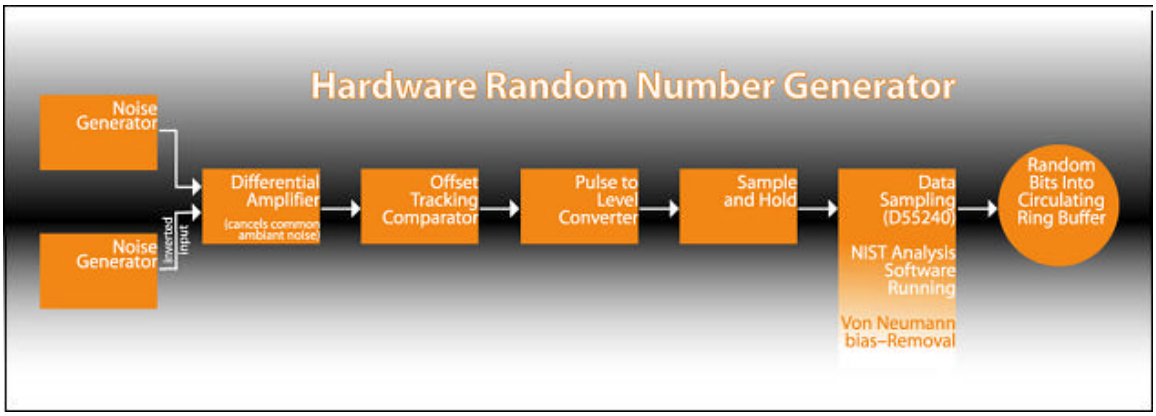
(c) *Encrypted Memory Bus:* The *DS5240* also contains an on-chip hardware encryption/decryption engine *(Triple-DES)* that operates at the same rate as the machine instruction stream. The crypto operation is performed on each program fetch. Therefore, data crucial to the cryptographic operation such as encryption keys and the controlling software are never seen outside the processor as plaintext.

[23] In addition, the *iButton* includes an in-package temperature sensor that detects sophisticated attacks that involve liquid nitrogen, helium and the like.

HARDWARE RANDOM NUMBER GENERATOR

The second integral component of the *Crypto 2000* board is a carefully designed and tested cryptographically secure *hardware random number generator* (RNG).

Far more secure than software techniques are *hardware-based* random number generators. Generating cryptographically secure random numbers in hardware is surprisingly hard[26] and is, in fact, more of an art than a science.



The great majority of crypto systems use off-the-shelf or even home grown *pseudo random* number generators that are just simple software programs that depend on an initial *seed* as a starting value. If you know the algorithm – by simply reverse engineering the software – and are able to ferret out or deduce the seed (or any subsequent seedlet), then the "random numbers" can be predicted.[24] Software generators are notoriously insecure[25].

Interestingly enough, a key factor to success is to keep the actual hardware random source as simple as possible and very understandable. Trying to create randomness by obfuscation *does not work.*

Systems occasionally attempt to use some seemingly "random event" such as mouse movements or the time between key strokes[27] or even strange effects such as minute airflow variances within disk drives to try to generate "randomness" to drive, in particular, key or key pair generation.

[24] Software generators also *repeat* at some point. There is, in fact, something more than just slightly perverse in trying to use the most precise machines known – *digital computers* – to generate random numbers. It does not make sense.

[25] E.g., see Knuth's *Seminumerical Algorithms* section 3.2-3.3.

[26] The primary problems are wrapped around issues of very high-frequency low-level analog signals, sampling theory, seepage of ambient noise, and, in general, the cussedness of such things.

[27] The well known original *PGP* encryption system developed by *Phil Zimmerman* used this method for key generation.

However, such methods are usually not practical in real world production applications – who wants to be required to wiggle a mouse whenever a secure key is to be generated – and who can guarantee that such methods are actually "random"?? Such "random events" are not known to be random at all.

The Crypto 2000 board produces random numbers by generating random electronic noise by known quantum physics processes and by then amplifying and sampling that noise[28]. The theory is quite simple – the practical working implementation is much more difficult.

It is impossible to prove theoretically that a process generates truly random numbers.[29] You can, however, convince yourself – and others – that a process generates random numbers by carefully examining and thoroughly understanding the underlying process itself[30] and then, and only then, verifying the results with comprehensive statistical tests.

[28] The hardware RNG uses two separate noise generators each of which employs two transistors. One transistor has its base-emitter junction reverse biased into the breakdown region which generates quantum random *current shot noise* that is fed into the other transistor which is configured as a normal common emitter configuration to act as a current to voltage converter. Negative feedback is employed to provide DC bias point stabilization and to minimize the effect of transistor component variations. The noise voltage is fed back to the reverse biased transistor to limit noise generation pulse width. The two random shot noise generators feed random pulses into a differential amplifier. The amplifier has two inputs – one inverts the incoming signal while the other does not. This property subtracts the signals from the two generators so that any signal components that are common to both – such as ambient electrical noise -- are canceled out to eliminate any external periodic interference that is introduced to the circuit via the power supply, ground bounce from the associated digital circuitry, or from EMI. A second operational amplifier is used as a ground generator to supply a virtual ground to the differential amplifier for proper operation. The conditioned random noise is then fed into an analog comparator that has its trigger reference derived by scaling and integrating its input signal to make an offset tracking comparator to quantize the analog noise. The offset is desired so the noise pulse rate is limited and the noise entropy is enhanced. The narrow quantized noise is then converted to digital 1's and 0's by simply clocking a JK flip flop with the quantized noise. A sample and hold made from another JK flip flop is used to sample and synchronize the random bit stream for processing by the DS5240 processor. The synchronized bit stream has any residual bias removed by using the classic *von Neumann* method and the stream of random bits is then injected into a circulating ring buffer that lives within the DS5240 processor.

[29] The most understandable definition of a random sequence that I know of is from *Exploring Randomness* by Gregory Chaitin (2001 Springer-Verlag ISBN 1-85233-417-7): *Something is random if it is algorithmically incompressible or irreducible.* Another good, but slightly less understandable definition is: *A member of a set of objects is random if it has the highest possible complexity that is possible within the set.* I prefer the first conceptual definition in this computational age of Turing, Gödel, Alonzo Church, et. al.

[30] For an excellent introduction to the history and subtle difficulties of building practical hardware RNGs, see *Terry Ritter's* web page at *http://www.ciphersbyritter.com.*

See *Appendix I* for results of the empirical tests that BSI2000 has run in the lab on a billion random bits generated by the Crypto 2000 *RNG*. A background software task running in the DS5240 samples, at a low rate, the quantum physics produced bits that are stored in a circulating ring buffer. The task continually also runs the *NIST 800-22 RNG* random number test suite[31] as a low priority background task to assure that some hardware fault has not caused bad bits to be produced. If generated bits begin to fail the suite, then an alerting message is sent to the associated TPU processor.

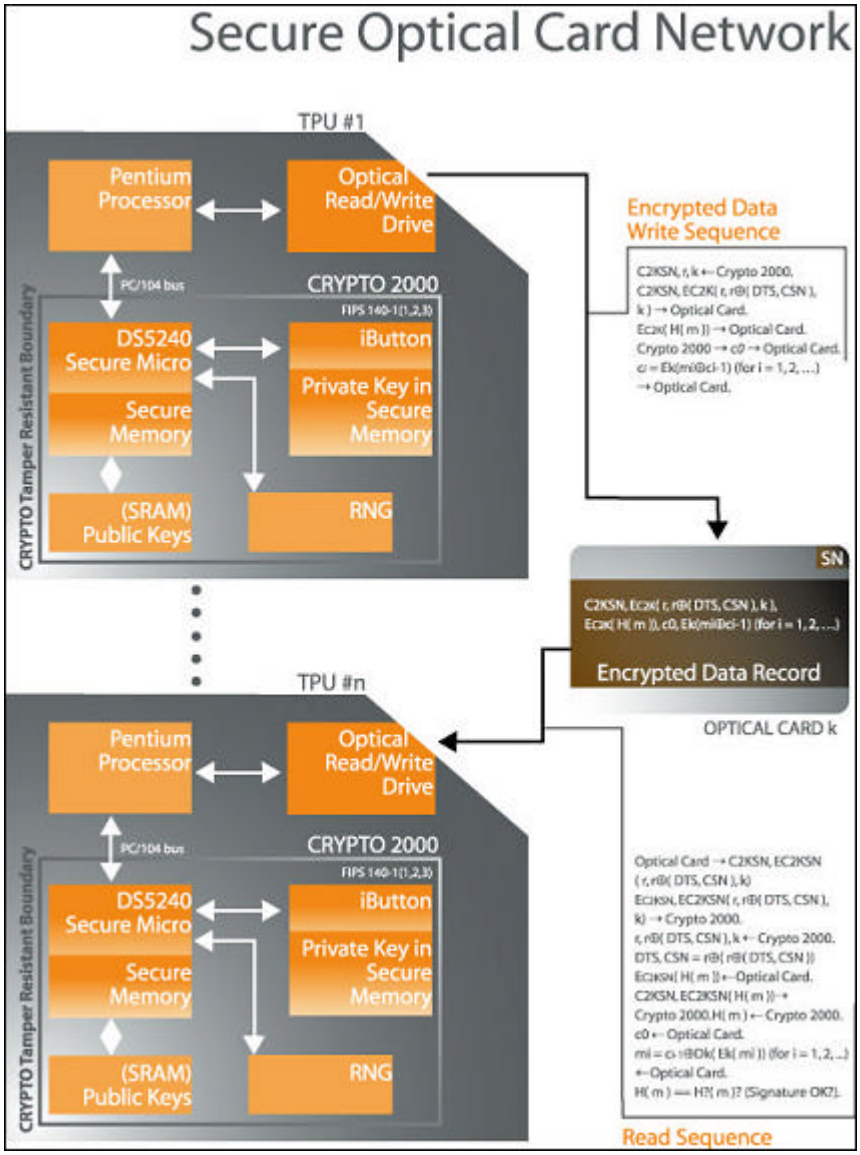
Coupling the above major components together – *a secure repository for secret keys, a secure key management system, and a cryptographically secure RNG* – all in a highly tamper-resistant package -- provides the underlying hardware support for the *SOCP Secure Optical Card Protocol*[32]

[31] http://csrc.nist.gov/rng/SP800-22b.pdf

[32] Commands that *Crypto 2000* responds to includes: Encrypt a record with the private key; Decrypt a record that was encrypted by another Crypto 2000 board; Generate, with the RNG, and return a 128-bit Session Key; Generate a 2,048-bit key pair and return the Public Key; Return the unique serial number of the Crypto 2000 board; and a few others.

SOCP CRYPTO PROTOCOL

A system of *n* transaction units (TPUs) with *k* optical cards appears as follows:



Each optical card TPU transaction unit consists of, at least, a read/write optical drive, an embedded Pentium processor, a human readable display (touch screen LCD), control buttons, interfaces to external or integral biometric devices, interface to external communication links if needed (Internet, LAN, etc), and a *specific daughter Crypto 2000 board*[33].

Realize that the terminology *public key*, as used here, does *not* mean that the key is actually public. It means that it is the decrypting key for a data object that was encrypted with the associated *private key*. No keys in the SOCP Protocol are actually "public" – *they are all protected.*

Symbols Defined for the Protocol:

All italicized symbols are defined within this section.

ASM -- The binary executable *Application Software Module* runs on the particular *TPU* Pentium. The *ASM* is encrypted as a Secure Record[34], is signed by the public key of the particular *Crypto 2000*, and is read from the *MBOC* at system startup time by the *SL*.

C2KD -- File of decrypting public keys for each *Crypto 2000* unit in a card system. *C2KD* is always encrypted with the private key of the specific daughter *Crypto 2000* and lives as $E_{C2K}(C2KD)$ in volatile memory of the *DS5240 Secure Processor* chip. Individual decrypting public keys are decrypted by the *FIPS 140-1 (1, 2, 3)* certified *iButton* on an as-needed basis and are *never* passed out of the Crypto 2000.

C2KSN -- Uniquely identifying serial number of a *Crypto 2000 unit* that writes a data record to an optical card. *C2KSN* is used by the receiving *Crypto 2000* to look up the decrypting key in the protected *C2KD* file.

Crypto 2000™ -- The secure key management and cryptographically secure hardware random number generator. The *Crypto 2000* board is contained within the *TPU* as an add-in security daughter module on the PC/104 bus and responds to commands from the controlling Pentium processor in the partner *TPU* only after it has received a correct copy of the 160-bit enable key[35] that is prestored in the secure memory of the *iButton* device. Usually, the enable key is *H(TPP)*. Each specific daughter *Crypto 2000* contains a 2,048-bit private key that resides in the nonvolatile protected memory of the iButton device. The private key never appears outside the *Crypto 2000.*

[33] The terminology *Specific Daughter Crypto 2000 Board* is used to mean the Crypto 2000 board that is specifically paired with and contained within a particular TPU. Each Crypto 2000 is loaded, when manufactured, with a unique serial number that identifies the unit and with unique and permanent crypto keys.

[34] Written with the *SOCP Write* protocol next defined.

[35] There are a lot of possible 160-bit sequences. 2^{160} (or about 10^{48}) such keys exist.

$D_{C2K}(c)$ -- The ciphertext *c* is decrypted with the public key of the specific daughter *Crypto 2000* to yield the original plaintext.

$D_k(c)$ -- The ciphertext *c* is symmetrically decrypted with the 128-bit random key *k*. *c* is the result of the original symmetric encryption with the key *k* where that key was generated by the hardware *RNG* on the *Crypto 2000*.

$E_{C2K}(m)$ -- The plaintext *m* is encrypted with the private key of the specific daughter *Crypto 2000* unit.

$E_{C2KSN}(m)$ -- The plaintext *m* is encrypted with the private key of a remote *Crypto 2000* that is identified by the uniquely identifying serial number *C2KSN* that is contained in the header block of the data record read from the optical card. The particular reading *Crypto 2000* can lookup the public decryption key of the remote *Crypto 2000* in its local copy of the protected *C2KD* file.

$E_k(m)$ -- The plaintext m is symmetrically encrypted with a 128-bit random key *k* where that key was generated by the hardware RNG on the Crypto 2000.

H(*m*) -- Data object *m* is hashed with the one-way cryptographically secure hash function (e.g., with the NIST 160-bit *SHA).*

$H_?(m)$ -- Verification to assure that a received data object is authentic – The question: *Does the calculated one-way hash value equal the hash that was stored on the card and then read?* is denoted by: $H_?(m) == H(m)?$

iButton™[36] -- The FIPS 140-1 (1, 2, 3) certified hardware *tough nut* that serves as the secure repository for the secret keys that are the primary basis of the security for the entire system. The *iButton* device stores the secret key for the particular *Crypto 2000* unit and is manufactured by *Dallas Semiconductor* and is designed for secure key storage.

k -- A random 128-bit encryption key that is used with a symmetric encryption algorithm. *k* is generated by the hardware *RNG* on the *Crypto 2000.*

MBOC -- The *Master Boot Optical Card* is read by the *SL* at system startup time and initializes the daughter *Crypto 2000* and the *TPU*. It is unique to each particular *TPU* and contains $E_{C2K}(C2KD)$ and $E_{C2K}(ASM)$ which are also signed by the specific daughter *Crypto 2000.*

RNG -- The cryptographically secure *Random Number Generator* is contained within the *Crypto 2000* board and is controlled and monitored by the *DS5240 Secure Processor*.

SL -- The *Secure Loader* comes to life from FLASH memory in the *TPU* at system startup. The *SL* first reads the Text Pass Phrase *TPP* from the machine operator to enable the daughter *Crypto 2000* to begin accepting and responding to commands from the TPU Pentium. *SL* then reads the $E_{C2K}(C2KD)$ and the $E_{C2K}(ASM)$ items from the *MBOC* at system startup and then, after signature verification, starts the *ASM* application on the Pentium.

[36] *iButton* is a trademark of *Dallas Semiconductor.*

TPP -- *Textual Pass Phrase* that is unique to each specific daughter *Crypto 2000*. The unique 160-bit *H(TPP)* unlocks the specific daughter *Crypto 2000*, if it matches the prestored value, at power up to enable it to respond to commands from the TPU Pentium processor.

TPU -- The optical card *Transaction Processing Unit* is available from BSI2000 (e.g., the SIGABA 2000 units sold to the United States *Immigration and Naturalization Service/Bureau of Citizenship and Immigration Services)*. A *TPU* includes a Pentium processor (1 Ghz), an optical card read/write device, an optional operator interface (touch screen LCD, keyboard, etc.), networking option, and a specific daughter *Crypto 2000* board – all in a hardshell metal industrial case.

The actual protocol is straightforward[37]

To Boot a Transaction Processing Unit (TPU) to Life:

(a) Power Up the TPU. The *Secure Loader* SL comes to life from FLASH memory on the embedded TPU Pentium processor.

(b) Enable the Crypto 2000 Board. After power up, the specific daughter Crypto 2000 will not respond to commands until it has been successfully passed the prestored correct 160-bit random bit-string that enables the board. The TPU human operator types in the Text Pass Phrase *TPP* (e.g., *"The time has come, the walrus said…"*[38]) to the *SL* which is one-way hashed to yield the 160-bit *H(TPP)* value. SL transmits *H(TPP)* to the Crypto 2000[39] to enable the board assuming that there is a correct match[40].

> *H(TPP) -» Crypto 2000 (Enable Board).*

(c) Read the Encrypted Set of all Public Keys. The *SL* reads the *Master Boot Optical Card* (MBOC) to initialize the key set in the Crypto 2000. The encrypted file $E_{C2K}(C2KD)$, that contains each Crypto 2000's decrypting public key, is read from the *MBOC* and transmitted to the Crypto 2000. The file was previously signed by the specific daughter Crypto 2000 as $E_{C2K}(H(E_{C2K}(C2KD)))$. The specific daughter Crypto 2000 is now ready to decrypt secure traffic received from optical cards that was securely written by any other TPU/Crypto 2000 pair in the network.

> $E_{C2K}(C2KD)$, $E_{C2K}(H(E_{C2K}(C2KD)))$ «- *MBOC.*
> $D_{C2K}(E_{C2K}(H (E_{C2K}(C2KD)))) == H_{?}(E_{C2K}(C2KD))$? *(Signature OK?)*
> $E_{C2K}(C2KD)$ -» *Crypto 2000.*

[37] Symbol usage: In the following, "a -» b" should be interpreted to mean that "a" is written to "b"; "a «- b" should be interpreted as "b" is written to "a". The symbol "¤" is the Exclusive-OR operator (XOR).

[38] Basic Information Theory shows that the TPP should be about twenty typical English words in length to yield a 160-bit "random" bit string. It should *not* be a literary phrase that would be susceptible to a dictionary attack (*Shakespeare, Lewis Carrol* and thousands of other texts are available on-line) but rather a phrase, with punctuation, that is easily remembered by the TPP owner.

[39] The first traffic on the PC/104 bus that interconnects the Crypto 2000 and the TPU Pentium is encrypted by the simple method of using the first 128-bits of the calculated H(TPP) as a symmetric session key. After communication is established, then a new session key is generated by the RNG and then used for some period of time until a new session key is produced and used.

[40] An alternative method would be to store the random 160-bit binary string, expected by the specific daughter Crypto 2000, on the MBOC. This method would be more convenient for the human TPU operator but would require physical security for the MBOC – which way to go depends on the specific security and operational needs of the application.

(d) Read, Verify, and Load the Application Software. The signed Application Software Module *ASM* is read as a secure record from the *MBOC* and is started on the Pentium processor of the TPU. *k* is the session key.

$E_{C2K}(k), E_k(ASM), E_{C2K}(H(ASM)) \text{ «- } MBOC.$
$D_{C2K}(E_{C2K}(H(ASM))) == H_?({}^{D}D_{C2K}(E_{C2K}(k))^{(E_k (ASM))})? \text{ (Sig OK?)}$
$\text{Pentium «- } {}^{D}D_{C2K}(E_{C2K}(k))^{(E_k (ASM))} \text{ (SL replaced with ASM on the Pentium).}$

Therefore, the complete startup sequence is:

$H(TPP) \text{ -» Crypto 2000 (Enable Board).}$
$E_{C2K}(C2KD), E_{C2K}(H(E_{C2K}(C2KD))) \text{ «- } MBOC.$
$D_{C2K}(E_{C2K}(H (E_{C2K}(C2KD)))) == H_?(E_{C2K}(C2KD))? \text{ (Signature OK?)}$
$E_{C2K}(C2KD) \text{ -» Crypto 2000.}$
$E_{C2K}(k), E_k(ASM), E_{C2K}(H(ASM)) \text{ «- } MBOC.$
$D_{C2K}(E_{C2K}(H(ASM))) == H_?({}^{D}D_{C2K}(E_{C2K}(k))^{(E_k (ASM))})? \text{ (Sig OK?)}$

$\text{Pentium «- } {}^{D}D_{C2K}(E_{C2K}(k))^{(E_k (ASM))} \text{ (SL replaced with ASM on the Pentium).}$

To Write a Secure Record to an Optical Card:

(a) Build the *Header Block*[41]*:* Ask the specific writing daughter Crypto 2000 unit for its uniquely identifying serial number *C2KSN*. Package the current date time stamp *DTS* and the uniquely identifying serial number of the target optical card *CSN* into a data record of length n-bits. The *DTS* and *CSN*[42] prevent *cloned records* and *block replay attacks.* Request the specific daughter Crypto 2000 unit to generate and return a random number, *r*, of length n-bits. Calculate *r¤(DTS, CSN)* which is used to blur the effectiveness of plaintext attacks where DTS and CSN would act as plaintext fodder[43]. Request the specific daughter Crypto 2000 to generate another random number, *k*, of length 128-bits to be used as the session key. Request the specific daughter Crypto 2000 to then encrypt, with its private key, the data record *(r, r¤(DTS, CSN), k)* and to then write that result, preceded by *C2KSN*, to the target optical card as the secure traffic header record.

> *C2KSN, r, k «- Crypto 2000.*
> *C2KSN,* E_{C2K}*(r, r¤(DTS, CSN), k) -» Optical Card.*

(b) Sign the plaintext: Calculate the one-way hash of the plaintext *m* and request the specific daughter Crypto 2000 to encrypt the 160-bit result with its private key. Write the result to the target optical card.

> E_{C2K}*(H(m)) -» Optical Card.*

[41] Certain unimportant implementation details such as a possible header tag, system version number, symmetrically encrypted record count, padding, and the like are eliminated here.

[42] The *card serial number* can be, if wished, not used. However, the protocol would then be more susceptible to *block replay attacks* but that is of no consequence to many applications. Fraudulently duplicated cards are, however, another issue.

[43] The *(r, r¤m)* method is used to make it much more difficult to launch an effective plaintext attack with *m* as the target. E_X*(r, r¤m)* is written to the card. *m* can, obviously, be recovered with the simple *r¤*D_X*(*E_X*(r, r¤m))*. The effect is to blur the plaintext.

(c) Encrypt the plaintext: Use a symmetric algorithm to encrypt the plaintext *m* with the 128-bit random key *k*. However, use *block chaining* to reduce the effectiveness of *plaintext* or *block repeat* attacks – Request the specific daughter Crypto 2000 to generate and return a 64-bit random number c_0 to be used as the *initialization vector* for the block chaining algorithm. Write the block chaining initialization vector to the target optical card and then write to the card the symmetrically encrypted 8-byte blocks of plaintext. The first block of the plaintext is m_1.

> *Crypto 2000 -» c_0 -» Optical Card.*
> *$c_i = E_k(m_i ¤ c_{i-1})$ (for i = 1, 2, …) -» Optical Card.*

Therefore, the complete secure record write sequence for the plaintext *m* is:

> *C2KSN, r, k «- Crypto 2000.*
> *C2KSN, E_{C2K}(r, r¤(DTS, CSN), k) -» Optical Card.*
> *E_{C2K}(H(m)) -» Optical Card.*
> *Crypto 2000 -» c_0 -» Optical Card.*
> *$c_i = E_k(m_i ¤ c_{i-1})$ (for i = 1, 2, …) -» Optical Card.*

Therefore, the complete secure record for the plaintext *m* is written to the optical card as:

> *C2KSN, E_{C2K}(r, r¤(DTS, CSN), k), E_{C2K}(H(m)), c_0, $E_k(m_i ¤ c_{i-1})$ (for i = 1, 2, …).*

To Read a Secure Record from an Optical Card:

(a) Read the *Header Block:* The uniquely identifying serial number, *C2KSN,* of the writing Crypto 2000 unit is the first item in the header record that is read from the card. The second item is $E_{C2KSN}(r, r¤(DTS, CSN), k)$. Request the specific reading daughter Crypto 2000 unit to decrypt the $E_{C2KSN}(r, r¤(DTS, CSN), k)$ record with the securely stored public key of the writing Crypto 2000 unit that is identified by looking up *C2KSN* in *C2KD.* The record $r, r¤(DTS, CSN), k$ is returned from the specific reading Crypto 2000. Calculate $r¤(r¤(DTS, CSN))$ to recover the date/time stamp *DTS* and the serial number of the target optical card *CSN.* Verify that *CSN* actually matches the serial number of the card being read[44]. *k* is the session key that was used to encrypt the plaintext.

> $C2KSN, E_{C2KSN}(r, r¤(DTS, CSN), k)$ «- *Optical Card.*
> $C2KSN, E_{C2KSN}(r, r¤(DTS, CSN), k)$ -» *Crypto 2000.*
> $r, r¤(DTS, CSN), k$ -» *Crypto 2000.*
> $DTS, CSN = r¤(r¤(DTS, CSN))$.

(b) Read the Authenticating Plaintext Signature: Read the $E_{C2KSN}(H(m))$ item and request the specific reading daughter Crypto 2000 unit to decrypt the item with the securely stored public key of the writing Crypto 2000 unit that is identified by looking up *C2KSN* in *C2KD.* Return the authenticating signature $H(m)$.

> $E_{C2KSN}(H(m))$ «- *Optical Card.*
> $C2KSN, E_{C2KSN}(H(m))$ -» *Crypto 2000.*
> $H(m)$ «- *Crypto 2000.*

(c) Read and Decrypt the Plaintext m: Read the c_0 block chaining initialization vector. Read and decrypt each of the $E_k(c_i)$ (for i = 1, 2, …) records with the symmetric algorithm. *k* is the symmetric session key.

> c_0 «- *Optical Card.*
> $c_i = m_i = c_{i-1}¤D_k(E_k(m_i)) (for\ i = 1, 2, …)$ «- *Optical Card.*

[44] If CSN does not match the actual card serial number, then a *block replay attack* is probably underway or a record has been cloned from another card and illicitly written to this card.

(d) Verify the Signature: Calculate the one-way hash of the just decrypted plaintext *m* and verify that it equals the previously decrypted *H(m)*.

$H(m) == H_?(m)$? (Signature OK?)

Therefore, the complete secure record read sequence for the plaintext *m* is:

$C2KSN, E_{C2KSN}(r, r¤(DTS, CSN), k)$ «- Optical Card.
$C2KSN, E_{C2KSN}(r, r¤(DTS, CSN), k)$ -» Crypto 2000.
$r, r¤(DTS, CSN), k$ «- Crypto 2000.
$DTS, CSN = r¤(r¤(DTS, CSN))$.
$E_{C2KSN}(H(m))$ «- Optical Card.
$C2KSN, E_{C2KSN}(H(m))$ -» Crypto 2000.
$H(m)$ «- Crypto 2000.
c_0 «- Optical Card.
$c_i = m_i = c_{i-1}¤D_k(E_k(m_i))$ (for i = 1, 2, …) «- Optical Card.
$H(m) == H_?(m)$? (Signature OK?).

ASSERTIONS

The author believes that the *SOCP* crypto protocol fulfills all of the original stated design requirements.

(a) The protocol, with the Crypto 2000 unit, is *more secure* than the level of security that is provided by the highest-level standard smart cards. The *fast* and *complete* zeroization of keys and other items and the several layers of physical tamper attack sensing provided by the *FIPS 140-1 (levels 1,2,3)* certified iButton and by the DS5240 Secure Processor that is the core of the Crypto 2000 module cause this assertion to be true.

(b) The one-way hash that implements a digital signature enables all records to be *authenticated, verified for integrity,* and *nonrepudiable.*

(c) *Known plaintext* and *dictionary attacks* are minimized with the data compression of the plaintext and by using the simple *(r, r? m)* method to blur plaintext fodder such as the date/time stamp and the serial number of the target optical card with, in essence, random strings that are then encrypted.

(d) *Man in the Middle* attacks are not effective due to the *digital signature authentication.*

(e) The protocol detects illicitly *cloned* optical cards. Each secure record contains, in encrypted form, the unique serial number of the original card to which it was written.

(f) If a TPU/Crypto 2000 unit is stolen and an attempt is made to reverse engineer and understand the system, the file of all Crypto 2000 public keys (always encrypted in volatile memory) and the secret key of the stolen TPU/Crypto 2000 would be exceedingly difficult to recover due to the protection provided by the iButton and the DS5240 *Tough Nuts.* To recover the private key would also entail *complete destruction* of the Crypto 2000 unit – *a standout event.* A periodic *Hot List* of missing or compromised TPU machines is sent, on optical cards, to each TPU as notification to ignore the *unclean* machines.

(g) The possibility of *Trojan Horse* attacks is minimized because the attacking software cannot obtain a copy of the 160-bit one-way hash of the *Text Pass Phrase* that is securely stored in the protected memory of the Crypto 2000 behind multiple layers of tamper resistant physical barriers. The specific Crypto 2000 unit *will not function at all* until it receives the correct and unique 160-bit enabling code from the TPU.

(h) If a Crypto 2000 unit is stolen, *it will not respond* to meaningful commands over the PC/104 bus until it has been activated with its unique 160-bit hashed Text Pass Phrase.

(i) Three separate pieces of hardware and the Text Pass Phrase must be stolen to have any realistic chance of a successful attack. The physical TPU with its specific daughter Crypto 2000 unit and the Master Boot Optical Card for that specific machine must be stolen to make any progress and even that is useless without the unique Text Pass Phrase. The Master Boot card should be stored separately from the TPU in a secure manner.

(j) The underlying hardware security of the protocol, with the Crypto 2000, is certified to *FIPS 140-1 (levels 1, 2, 3).*

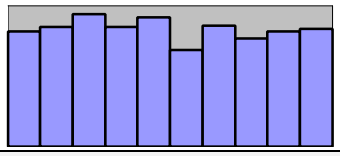
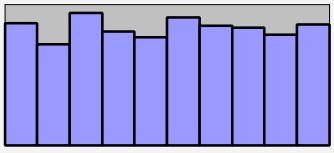
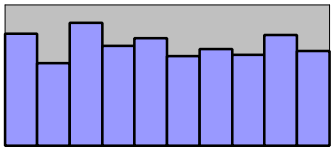
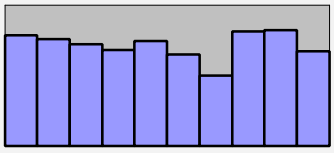
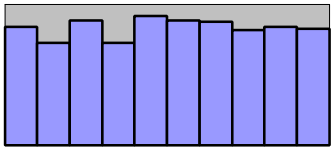
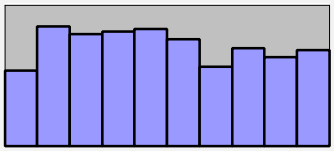
USE IN THE REAL WORLD.

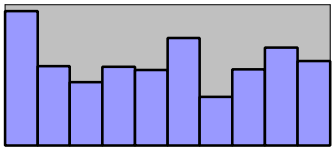
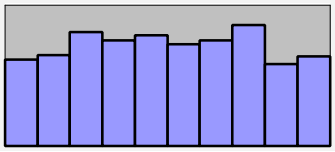
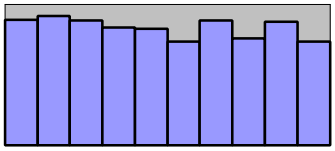
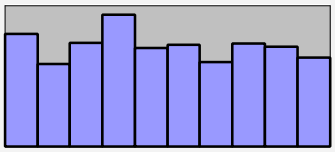
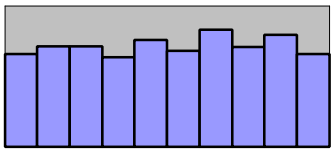
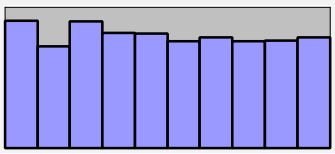
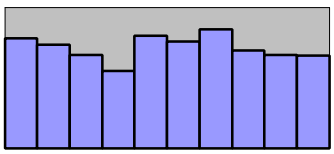
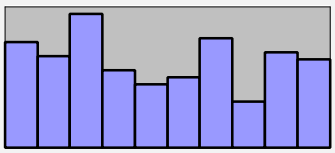
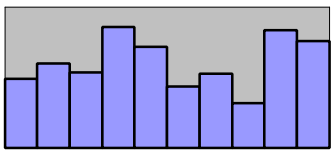
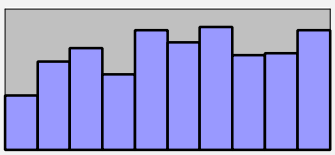
A system that uses *Crypto 2000* boards running under the *SOCP Secure Optical Card Protocol* is scheduled to be used first in an optical financial card project in South Africa.

Statistical Evaluation of the CRYPTO 2000 RNG (Appendix I)
Jay Miller, BSI2000, Inc. 3.13.2002

We have tested a total of 1,000,000,000 bits of output from the Crypto2000 RNG over the course of 1,000 independent trials to verify the hypothesis that the output is as random as the underlying quantum physics. The results of these tests, performed with the NIST 800-22 RNG test suite, are shown below. The NIST publication recommends two approaches for interpreting the findings: 1) examine the proportion of successes versus failures and 2) examine the distribution of results with some expectation of uniformity. A detailed assessment follows in the Outcome Analysis section.

Test Results

1	Monobit Frequency	985:15 (0.985)		0.655854	**Pass**
					
3	Runs	985:15 (0.985)		0.140453	**Pass**
					
5	Binary Matrix Rank	993:7 (0.993)		0.796268	**Pass**
					

7	Nonperiodic Template[ii]	146488:1512 (0.990)		0.041723	**Pass**
					
9	Universal Statistic[iv]	987:13 (0.987)		0.723804	**Pass**
					
11	Linear Complexity[v]	981:19 (0.981)		0.649612	**Pass**
					
13	Approximate Entropy[vii]	985:15 (0.985)		0.165340	**Pass**
					
15	Random Excursions 1	4877:51 (0.990)		0.489224	**Pass**
					

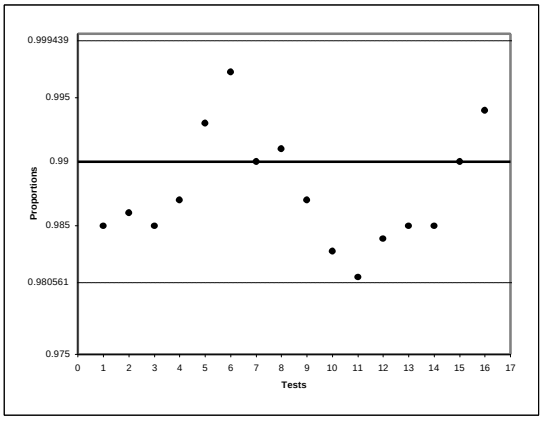
Outcome Analysis

Each of the tests in the suite outputs a *P-value*. This value summarizes the evidence for our hypothesis or, more precisely, the probability that a perfect RNG would have produced data less random than the data tested. The possibility of erroneously rejecting our hypothesis is called the *significance level*, denoted 'α'. For this test, then, α is the probability of a nonrandom conclusion despite truly random data. Comparing the output of the test, a P-value, to α decides whether the particular test passes or fails. Typical values for α in cryptographic applications are about 0.01.

We now consider the above results in light of the recommended interpretation approaches.

1) **Pass:Fail Proportions.** Given a significance level >0, we expect a certain proportion of successes and failures. Too few passes indicates the data exhibits 'positive' weaknesses (*e.*g. patterns). Too few failures indicate the reverse. That is, if an attacker knows a certain bit stream will *never* fail certain tests, she has increased chances of guessing its output.

   To decide if the results lie within an acceptable range, we define a confidence interval by estimating a true standard deviation. This interval is therefore based on sample size (m = 1,000) and significance level (α = .01): $? 3\sqrt{\frac{? (1 ? ?)}{m}}$. Plugging in the correct values results in a range of ?0.009439. We illustrate this confidence interval by plotting each test's proportion. The dotted lines in the graph show the upper and lower bounds of the confidence interval.



   Since all of the plots fall within the specified confidence interval, the proportion of passed tests to failed tests indicates our hypothesis is correct.

2) **Uniformity in P-values.** For each test above, its P-values have been categorized into one of ten equal intervals between 0 and 1. The histograms shown illustrate the quantity of P-values per interval in order to provide visual evidence of uniformity.

   Further examination is possible by computing an overall P-value *of the P-values* using a Goodness-of-Fit Distributional Test. We must first calculate $? ^2 ? \sum_{i?1}^{10} \frac{(F_i ? s/10)^2}{s/10}$, where F_i is the number of P-values in interval i and s is the total number of P-values. The overall P-value P_o is then calculated using the upper incomplete gamma function: $P_o ? ?(9/2, ? ^2/2)$. If the resulting value is ?0.0001, we consider the sequence uniformly distributed. As the table shows, all values of P_o lie above the 0.0001 threshold, confirming our visual evidence and indicating our hypothesis is correct.

Conclusion

Since both interpretation approaches indicate our hypothesis is correct, we conclude the output of the Crypto2000 RNG is as random as the underlying quantum physics.

References

Rukhim, et al. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST Special Publication 800-22, May 15, 2001. Available: http://csrc.nist.gov/rng/.

[i] Block size (M) = 20,000.
[ii] Template length (m) = 10.
[iii] Template length (m) = 10.
[iv] Block size (L) = 12; initialization steps (Q) = 40,960.
[v] Block size (M) = 1,000.
[vi] Block size (m) = 2.
[vii] Block size (m) = 2.