Turnstile -- A Novel Encryption Scheme

M. Weir[1]

1: Ampex Data Systems Corporation, Hayward, California, USA

Abstract: The need for encryption throughout the telemetry industry has become ubiquitous. With commercial and governmental interests to protect, both data-at-rest (DAR) and data-in-transit (DIT) is now essential.

However, encryption is like seat belts and crash helmets: a nuisance until that moment when you really need the protection! The challenge for effective encryption, then, is a management scheme for the encryption keys that imposes as small of an additional workload on the operations staff.

Turnstile is an encryption scheme that combines tried, tested and trusted algorithms in such a way as to make recordings inaccessible even if an adversary has full access to, and control of, the test article. This allows engineers to configure a system with full encryption capabilities without having to "re-key" the unit regularly.

By employing existing algorithms, Turnstile benefits from the quality and trustworthiness of those algorithms and certifications (such as FIPS 140 or Common Criteria). Turnstile can be applied both to data-at-rest and data-in-transit applications with small variations.

Keywords: Cyber Security, Encryption, Test Data Acquisition

1. Introduction

There are few fields of data processing, be it industrial, commercial or consumer, where the idea of encryption of data at rest is not anticipated or expected. However, even when the capability is present, it is frequently not engaged.

This paper explores the reasons why data at rest encryption is not as widely used as one might hope or expect, and thus the context behind the Turnstile Encryption Scheme developed by Ampex Data Systems Corporation specifically to help address those issues and make data at rest encryption a less intrusive technology with the required levels of security and integrity.

Note: to the extent Ampex has intellectual property rights in the Turnstile scheme, RAND licenses are available.

2. The Problem Space

An assessment of any variety of data protection approach fundamentally must begin with characterizing what, precisely, the threats *are*, and how a specific approach might acceptably address those threats.

For data-at-rest encryption applications, the objective of the encryption is obvious: the technology should prevent unauthorized access to the stored data. But there are multiple scenarios in which the threat of unapproved individuals getting to that data might arise, and not all of them are applicable to every project, nor are they equally probable or do the result in the same consequences is the access is obtained.

The unauthorized access attempt could be accidental or malicious, targeted, or opportunistic. In general, accidental access can be benign (e.g., a fellow employee getting access to data they should not have been able to see), but of course it is entirely possible that an accidental leak might then be subsequently exploited by a malicious actor.

When it comes to malicious attacks, the typical perpetrator may be seeking competitive advantage commercially, or even on the national level. And that creates a necessity for protection both for truly sensitive data as well as for data that can be used to assemble information of value.

And of course, with the general consolidation of the aerospace industry, the line between military and civilian vehicles is less distinct than it once was. This has the net effect of spreading attacks; for example, an unfriendly nation state may design an attack against what they perceive as a military program, but which is actually seen as a civilian platform by its developer, and therefore potentially less well protected.

These scenarios are described below:

2.1 Total Loss of Control.

When data at rest protection is discussed, it is usually this sort of threat that people consider: an entire operational system, including on-board storage, is lost.

In this situation, all the hardware and software that's needed to access the data is available; the system was working normally prior to loss, and so the design of the data at rest solution, with particular focus on key retention, is the barrier protecting the data!

2.2 Data at Rest in Transit.

While this may seem like a contradiction in terms, this scenario covers situations where media is in transit between the recording system and the exploitation system.

The key factor in this scenario, unlike the previous one is that the media is "bare": to access the sensitive information, additional hardware will be required. With commodity media, this may not present an adversary with much, if any, of a challenge, but this situation represents the most prevalent one, as it includes lost, abandoned, or discarded media.

2.3 Recorder System Information Assurance Protection.

In order to protect the integrity of a subsystem, it is necessary to protect all points of entry. While it may not be a particularly obvious vulnerability, an adversary can use removable media as a mechanism to transport malware to a system (in much the same way as attackers use removable media – CDs and thumb-drives, for example – to infect PCs).

This can be prevented by encrypting the media. If a host (i.e., the on-vehicle recorder) refuses to mount unencrypted storage, and the keys for the encryption are well protected, then only trusted parties can use that media to transfer data.

2.4 Data Validation and Authentication.

This scenario is the reverse of the previous one: if only systems that have the (protected) key can mount the media, then it can be asserted that the media contains valid, authenticated data that cannot have been forged.

This provides a useful "chain of custody" mechanism for data being used for official purposes (e.g., criminal prosecution).

3. Key Handling for Encryption on Test Vehicles

With the widespread standardization of encryption algorithms (see below), the process of encrypting the large amounts of data involved ("bulk data") is pretty much a solved problem. But the challenge of managing keys – typically loading, storing and protecting them, but also generating and destroying them remains; it is the handling of keys that is the major hurdle in designing and applying data-at-rest encryption.

The simplest form of key handling is simply to provide a channel through which an operator is expected to send the keying information. This channel may be something like a TCP/IP socket over Ethernet, or as simple as a serial port; the distinguishing feature here is that the host system (i.e., the system trying to unlock the encrypted data-at-rest media) will suspend operations pending the delivery of the key.

Obviously, in some way this just pushes the problem out of host system: instead of the host unit needing to address secure key storage, whatever it is that provides the keys through the channel must achieve the same functionality.

It should be noted that any key handling channel may pose a security vulnerability of some kind; this point is usually addressed by either having encrypted channels (e.g., by using a network VPN solution) or by "wrapping" keys, i.e., encrypting the key itself, or both. The passwords or keys for those operations introduce their own key handling issues, but in general no new *types* of challenges.

The next type of key handling is where the host system contacts another system on the vehicle and requests the keys from there (often termed the "root of trust"). This request would be issued every time the keys are required. The difference between the first scenario and this one is that authentication of the host to the key store will almost inevitably be required, as any store will need confidence that it's not handing out sensitive information (the keys) to the wrong client!

Again, this is another example of pushing the problem elsewhere, but architecturally having a centralized secure key store is often required or desired in an overall system-of-systems design.

A variation on this scheme is the use of key dongles, sometimes called crypto ignition keys (CIKs): a physical item is plugged into the host system and the keying material is read off the dongle. So instead of a centralized key store, you have a localized one. Dongles / CIKs tend to use simple serial protocols (SPI or I^2C, for example) or USB, and are entirely dependent on the host system pulling the data out of them.

However it is that the keys get transferred to the host system, two additional capabilities frequently are required: local storage, and destruction.

The simplest approach is to store the key in volatile memory (RAM); when power fails, the key disappears. The complication lies in cases where it is necessary for the key to survive power cycles – either to recycle a potentially misbehaving system, or as a characteristic of the vehicle (e.g., switching from one generator/alternator to another) or as part of a normal CONOPs where a vehicle is prepared significantly prior to an operation, and then gets shutdown until start time.

These all dictate some kind of power hold-up circuitry, or use of non-volatile memory, or both.

For power hold-up, the obvious issue is the duration required. "Short" durations can use supercapacitors, which have the significant benefit that they *will* get exhausted, at which point there will be no power to preserve the sensitive keying information, so it will evaporate as soon as power fails.

For longer durations, supercapacitors become impractical: while it's certainly possible to build an arbitrarily large capacitor bank, the process of charging complicates the design: one must limit inrush current to within acceptable bounds, and additionally calculate the relationship between charge time and hold-up capability.

The alternative to supercapacitors is of course a battery. Even surmounting the (legitimate) bias against batteries for maintenance and safety reasons, batteries add an additional complication: an active watchdog of some kind is required to ensure that the key doesn't last *too* long, because unless one's using rechargeable cells (with an even stronger legitimate bias against), the battery has to be sized to support the application across multiple power failures; once the battery is exhausted, the system is impaired!

Of course, the challenge can be made easier by splitting the recording subsystem in two: the recorder proper, and a much smaller co-processor that acts as a key store; so long as power is maintained to the key store, the keying information can be preserved without having to power the whole recorder. Typical key store systems might be based on a microcontroller device, using common chip-to-chip interconnects like I^2C, RS-232/422, or USB.

The alternative to storing keying information in volatile memory is, of course, to use non-volatile storage. The obvious problem with this is that the if the recording device fails (e.g., is involved in a crash) then the non-volatile memory will retain everything needed to extract the encrypted, protected information, thereby defeating the whole purpose of encryption in the first place!

To avoid this problem, the obvious solution is to use "secure" storage devices, like Trusted Platform Modules (TPM). The concept behind these devices is that they are secure crypto-processors that help with actions such as generating, storing, and limiting the use of cryptographic keys. Many TPMs include multiple physical security mechanisms to make it tamper resistant, and malicious software is unable to tamper with the security functions of the TPM.

Unfortunately, the drawback to the use of these sorts of coprocessor device is that they have to be responsive to the host system, so that when the host starts up, it can extract the necessary keys to unlock the media. The "risk window" can be narrowed by time-based criteria, under which the crypto-processor purges itself once a given interval has elapsed. Of course, if the reference time is derived from a regular system clock, that can be defeated by simply resetting the date and time, so a secure system would typically use a non-resettable counter circuit to trigger the key purge when a particular value is reached.

In summary, protecting keys so that they survive power cycles but are not vulnerable to exfiltration in such a way as to defeat the purpose of encryption is a hard challenge to solve. It would be better, then, to have a sort of "write only" storage device, where the data cannot be recovered in the field, only back at base!

4. Encryption Technology

Encryption can be broken into two types of algorithm: those which use the same key for both reading and writing ("symmetric" algorithms), and those that use different keys to encrypt and decrypt ("asymmetric" algorithms).

For symmetric encryption, also known as "block ciphers", as of 2023 this is almost exclusively the Advanced Encryption Standard (AES), codified by FIPS 197 [1].

The reasons for this near-universal adoption are clear: AES is a published algorithm developed the Belgian cryptographers Vincent Rijmen and Joan Daemen, which was adopted by the US National Institute for Science and Technology (NIST) in 2001. As such, it has been validated by the global community's scrutiny of the algorithm's mathematical underpinnings, and a given implementation should be certified by the NIST Cryptographic Algorithm Validation Program (CAVP) and is therefore "correct".

For asymmetric encryption (those using different encrypt and decrypt keys), there's a split between the venerable RSA algorithm (named after its creators, Messrs. Rivest, Shamir, and Adleman), and Elliptical Curve Cryptography (ECC); while the mathematics underpinning the two are very different, they can be considered functionally interchangeable in most cases. These too are codified by standards (FIPS 186 *"Digital Signature Standard"* [2]) and validated by the global community. Formal NIST certification tends to be wrapped up with the functionality for which these technologies are most often used ("key exchange" and identity verification), but a broader-scope certification program like FIPS 140[1] "*Security Requirements for Cryptographic Modules"* [3] will also certify the asymmetric algorithm(s).

The advantage of asymmetric keys for test vehicle applications (and others) is clear: by loading only the "encryption key" (and not the "decryption key") on board, the system cannot recover data that has been written. However, there are challenges…

5. Encryption in Operation

Whichever type of encryption (symmetric or asymmetric) is used, the basic operations are the same: chunks of data are fed into the algorithm and the enciphered output emerges. However, the size of those chunks is limited: for AES the chunks are always 16 bytes (128 bits) long, while for the asymmetric variety the chunk length to be encrypted should not exceed the key size.

This leads to what can be referred to as the "repeating data issue": encrypting two chunks with the same key will result in the same output. And since the chunks are quite small, the situation shown in **Figure 1** will inevitably occur.

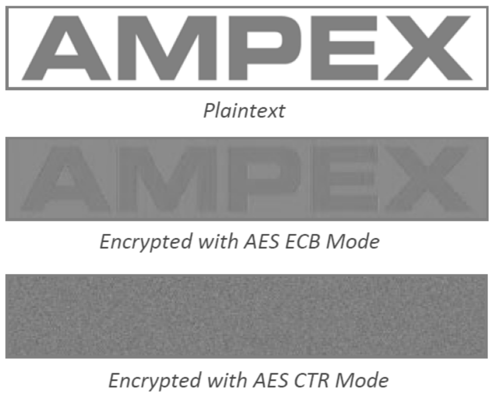

Figure 1 Repeating Block Issue

Figure 1 shows two AES "modes" defined by NIST as SP800-38 *"Recommendation for Block Cipher Modes of Operation"* [4]; the first, Electronic Code Book (ECB) mode, simply repeatedly encrypts each 16-byte block with the same key, and is obviously unsuited to plaintext with

[1] The standard is generally known as FIPS 140. The number after the dash is the edition number. FIPS 140-3 superseded FIPS 140-2 in March 2019.

repeating patterns. The second, Counter (CTR) mode, splits the message into a numbered succession of blocks, and merges the encrypted version of the plaintext with an encrypted version of the numbered counter, and in so doing changes the effective key being used to encrypt each block of the plaintext.

An alternative to Counter mode, the Galois/Counter mode (defined by NIST SP800-38D [5]) can be used, which adds a extra layer of protection against corruption of the recording or tampering, although it should be noted that tampering with an encrypted file can often be easily detected anyway, as the alternations can only be made "blind" and so will likely not align with e.g., packet boundaries in the encrypted data.

Unfortunately, the approach of using these sorts of modes to vary the "effective key" for each block cannot work for asymmetric keys, as each potential "effective encryption key" must have a related decryption key, and the relationship between the two is, by definition, mathematically "extremely hard" to derive.

To summarize, then, symmetric algorithms (i.e., AES) with appropriate "modes" work well large volumes of data but suffer from the problem that if a system has the encryption key, they also have the key needed to decrypt. And on the other hand, while asymmetric keys are an elegant solution to the key handling challenges, they are not much use for applications where more than a few bytes must be encrypted.

6. Turnstile

The concept behind the Turnstile Encryption System is to use both a symmetric and an asymmetric algorithm in concert.

Turnstile is designed for file or stream encryption, as opposed to "full disk" encryption. This permits the use of multiple keys and algorithms. Turnstile is also not suited to applications with huge numbers of files or "read-mostly" systems like NAS devices, since (a) each file (or small group of files) is separately encrypted, and (b) by design, Turnstile prevents the reading of a file.

Turnstile doesn't implement the encryption operations *per se*, relying instead on other implementations, which allows the use of FIPS 140 and/or Common Criteria certified solutions. Instead, Turnstile manages creates and Ephemeral File Encryption Keys (EFEKs), which are then used for bulk data encryption. In other words, it could be argued that Turnstile operates *above* FIPS 140, without reducing the "goodness" that a FIPS 140 implementation provides.

That bulk data encryption process will be (typically) be performed by AES with a suitable mode such as Counter mode using an EFEK. The EFEK is then encrypted with an asymmetric algorithm (typically ECC, but possibly RSA), and stored alongside the bulk data files.

Turnstile functions by generating a key pair for use with the asymmetric algorithm. One half of the pair, dubbed the "write key" or the "public key" (after it's similarity to Public Key Infrastructure schemes) is transferred to the test article, while the other half, the "read" or "private" key, is retained securely, for example in the control room. The "write" key is not particularly sensitive: an adversary posing that key could possibly create forged data files but could not manipulate or access the data in authentic ones.

6.1 Random Number Generation.

The strength of the encryption on the bulk data files comes down to the strength of the EFEK, so it is essential that Turnstile use a robust and well-understood approach to the EFEK creation process. As with all good keys, the EFEKs are random bit strings (typically, a 256 bit string of random "0"s and "1"s).

Perhaps paradoxically, cryptographically random bit generation (RBG) usually involves two elements: a source of entropy, often referred to as a True Random Number Generator (TRNG) and a Deterministic Random Bit Generator (DRBG), also known as pseudo-random number generators.

The TRNGs typically employed by encryption systems are typically implemented by electronics phenomena, such as thermal noise.

One might ask why the DRBG is required if one has a TRNG? The simple answer is that TRNG implementations tend to be limited in the number of random bits per second that can be produced. The use of a DRBG thus "smooths" the output of the TRNG so that bursts of requests for random numbers (by applications, in this case Turnstile) don't overload the TRNG and skew the randomness of the output. It is also true that proving the randomness of a TRNG across all potential operating conditions is hard, so using a DRBG provides slightly greater confidence across boundary conditions, such as temperature or power supply brownouts.

The approved methods for the use of entropy sources and the DRBG algorithms are codified in NIST SP 800-90A Rev.1 *"Recommendation for Random Number Generation Using Deterministic Random Bit Generators"* [6], SP 800-90B *"Recommendation for the Entropy Sources Used for Random Bit Generation"* [7] and the draft SP 800-90C *"Recommendation for Random Bit Generator (RBG) Constructions (3rd Draft)"* [8].

6.2 EFEK Storage.

Once the EFEK has been generated, it is encrypted using the "write" / "private" key. Typically, there will be some additional data encrypted alongside the EFEK, which provides different types of assurance, such as detecting when the system is booted in order to flag unexpected resets that might indicate tampering.

The encrypted EFEK and that additional data, known collectively as the Encrypted Key Control Block (EKCB) is then stored alongside the bulk data file that is encrypted with that EFEK.

The method, and indeed the location, by which the EKCB is stored is unimportant: it could be prepended to the bulk data file, it could be written in a separate "sidecar" file, saved in a database or even written to separate media. The only requirement is that the EKCB data be associated with the bulk data file, which is straightforward with prepending the data, but requires a little extra work with the other methods.

Once the bulk data file is completed (i.e., when the file is closed), the EFEK is purged for memory. The only place it can be obtained is now from the EKCB.

For best security, the system will periodically close a recording file, discard the EFEK, and create a new file with a new EFEK and EKCB.

6.3 Decrypting the Bulk Data Files.

Extracting the data from the recording files requires first recovering the EFEK from the EKCB. This requires the "read" or "private" key.

Since that key is simply not loaded on to the test article, an adversary taking control of the recording system just doesn't have the necessary pieces to decrypt the data files. As long as the asymmetric algorithm and the read key are secure, the bulk data is secure.

6.4 Turnstile Applications for Streaming.

Up to this point, this paper has been considering the challenges of encryption mostly from a data-at-rest perspective. However, Turnstile can be equally relevant to streaming telemetry.

The process of generating the asymmetric key pair and the EFEKs is identical to that used for data-at-rest applications, as is the concept of the EKCB. Obviously, instead of writing files encrypted with the EFEK, the telemetry stream is encrypted and transmitted, but otherwise the design mirrors the data-at-rest approach.

The difference lies in what the test article does with the EKCB: rather than storing it, it must be transmitted "alongside" the streaming data. This can be accomplished using either a packetization scheme on the data link, using a header bit to signify whether the data is an EKCB or encrypted data, or an out-of-band channel operating in parallel.

7. Conclusion

Key handling for data encryption is a hard problem to solve when controlled key persistence is required. Turnstile simplifies the process by creating a scheme in which the only keying data stored on a vehicle is not sensitive, because it cannot be used to decrypt previously written data.

Turnstile doesn't work for all applications, but for video and instrumentation recordings ("Chapter 10" and the like), it provides an elegant solution for secure protection of sensitive data.

8. References

[1] National Institute for Standards and Technology, FIPS 197 Advanced Encryption Standards, 2001. https://csrc.nist.gov/publications/detail/fips/197/final

[2] National Institute for Standards and Technology, FIPS 186-5 Digital Signature Standard, 2023. https://csrc.nist.gov/publications/detail/fips/186/5/final

[3] National Institute for Standards and Technology, FIPS 140-3 Security Standards for Cryptographic Modules, 2019. https://csrc.nist.gov/publications/detail/fips/140/3/final

[4] National Institute for Standards and Technology, SP 800-38A, Recommendation for Block Cipher Modes of Operation: Methods and Techniques, 2001. https://csrc.nist.gov/publications/detail/sp/800-38a/final

[5] National Institute for Standards and Technology, SP 800-38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, 2007. https://csrc.nist.gov/publications/detail/sp/800-38d/final

[6] National Institute for Standards and Technology, SP 800-90A Rev. 1, Recommendation for Random Number Generation Using Deterministic Random Bit Generators, 2015. https://csrc.nist.gov/publications/detail/sp/800-90a/rev-1/final

[7] National Institute for Standards and Technology, SP 800-90B, Recommendation for the Entropy Sources Used for Random Bit Generation, 2018. https://csrc.nist.gov/publications/detail/sp/800-90b/final

[8] National Institute for Standards and Technology, SP 800-90C (Draft), Recommendation for Random Bit Generator (RBG) Constructions (3rd Draft), 2022. https://csrc.nist.gov/publications/detail/sp/800-90c/draft

9. Glossary

AES:	Advanced Encryption Standard
CAVP:	Cryptographic Algorithm Validation Program
CIK:	Crypto Ignition Key
CMVP:	Cryptographic Module Verification Program
CTR:	Counter
DAR:	Data at Rest
DIT:	Data in Transit
DRBG:	Deterministic Random Bit Generation
ECB:	Electronic Code Book
ECC:	Elliptical Curve Cryptography
EFEK:	Ephemeral File Encryption Key
EKCB:	Encrypted Key Control Block
FIPS:	Federal Information Processing Standard
I^2C:	Inter-Integrated Circuit
NIST:	National Institute for Science and Technology
RAM:	Random Access Memory
RAND:	Reasonable and Non-Discriminatory
RBG:	Random Bit Generator
SPI:	Serial Peripheral Interface
TPM:	Trusted Platform Modules
TRNG:	True Random Number Generator
USB:	Universal Serial Bus
VPN:	Virtual Private Network