Analysis of IBM Custom Key Block formats to PCI PIN Security Requirements

UL report on the compliance to key block format requirements contained within the PCI PIN Security Requirements document and associated FAQs.

Reference Standard:	PCI PIN Security Requirements
Certification Body:	PCI Security Standards Council LLC
Evaluation Report No:	UL13487133
Authors:	Andrew Jamieson
	Joehannes Bauer
	Sajal Islam
	Benoit Feix

16 September 2022

Revision history

Revision	Date	Nature of amendment
0.1	26 Oct 2020	Initial creation and release
1.1	28 Oct 2020	Update after customer review
2.2	22 Dec 2020	Update to include VLS key block
2.3	1 Feb 2021	Update to include key block examples
2.4	3 Feb 2021	Update to include VLS key block example
2.5	19 Mar 2021	Update after customer review
2.6	22 Mar 2021	Final update and release
2.7	5 May 2021	Update to include summary of VLS
2.8	6 May 2021	Update list numbering typo
2.9	16 Sep 2022	Add appendix with bios of the authors
2.9		

Executive Summary

PCI SSC manage rules for the use of cryptographic keys within payment systems, particularly those used to secure and process PIN based transactions. The requirements for the key management of such PIN security related keys are provided in the PCI PIN Security Requirements, and these rules now mandate the use of 'key blocks' within the local environments of payment service providers. Mandates for the use of key blocks with external connections are coming at later dates within this decade.

The requirements for key blocks are set to refer to existing standards of TR-31 and ISO20038. Some technology providers have implemented key block systems that are not identical to those outlined in these standards, and PCI require that an independent review of these implementations is performed to validate they provide equivalence, with specific rules set to define the interpretation of this equivalence.

IBM have contracted UL to produce such a report on one of their key block implementations. This report is produced based on documentation provided by IBM, without any functional testing.

Through this review, UL has concluded that the WRAPENH3 and VLS formats designed by IBM can be considered equivalent to the standards of TR-31 and ISO20038, within the scope of 'equivalence' as put forward by PCI SSC for the purpose of such a review.

Table of Contents

Payment Key Management Overview

The financial industry can be considered one of the earliest widespread adopters of encryption systems, outside the military and governmental arenas. Methods to encrypt customer PINs were standardized in ISO9564 from 1991[i], with general principles of key management outlined in ISO11568 from 1994[ii]. Indeed, in many geographies, the use of encryption predates these International Standards by a decade or more.

Of course, the secure use of encryption relies upon the secure storage and management of the cryptographic keys used for these operations. This lead to the creation of even further standards for secure cryptographic devices, in documents such as FIPS140-1[iii] (published in 1994, now superseded by FIPS140-3) and a more payments focused ISO13491 (published in 1998[iv]).

In these standards, and the implementations that they both preceded and were informed by, the ways in which cryptographic keys may be generated, loaded, used, and replaced within payment instruments were developed. These processes are collectively understood as the 'life cycle' of the cryptographic keys that are used, and are governed by key management practices. As payment systems security has matured, the use of cryptography has been expanded from purely encrypting PINs to protecting other data (such as the payment card data itself), as well as providing authentication across message transmissions, and providing for the update of the working keys stored in a payment acceptance device.

In payment systems, it is common to consider there are three high level forms of key management:

1) Fixed
2) Derived Unique per transaction
3) Master/Session

In a Fixed Key system, a single key (for each purpose) is loaded into the payment acceptance system, and this is used throughout the lifetime of the product. A fixed key is often never changed, it is *expected* to not be changed, as it requires a physical process for any re-keying operation (although of course the keys are unique per purpose and device). Regardless of how long the payment acceptance system is in operation, or how many transactions are performed on this system, the keys used remain identical.

In a Derived Unique per transaction key management system, a single key is loaded into the payment acceptance device and new keys are generated for each transaction that is performed. This is achieved through the use of an on-device key derivation process, which uses a pseudo-random process for generating a new key from the previous key data, potentially along with additional transaction related data. The most commonly implemented Derived Unique key management system is that standardized in Annex C of ANSI X9.24 (known as Derived Unique Key Per Transaction, or DUKPT), although other methods do exist and are used around the world.

The key derivation method used in a Derived Unique key management system will often implement a separate process for host-side key derivation, allowing for easy derivation of any arbitrary transaction keys at the host whilst maintaining forward secrecy over key derivation at the acceptance terminal side. This alleviates the risk of the terminal/host key sets becoming out of sync. The derivation process will also generally allow for the creation of more than one

key, so that there may be different transaction unique keys for each purpose required during the transaction.

The third common form of key management used in payment systems is Master/Session key management, which is essentially a catch-all phrase used for any key management process which is neither Fixed Key, nor Derived Unique. This form of key management allows for the loading of new keys as required, although key derivation may also be included as part of this process. As implied by the name, a Master/Session key management system will result in the implementation of some form of 'key hierarchy', where the lowest level keys – the 'working' or 'session' keys – are loaded into the payment acceptance system using other keys that are higher-up in the chain.

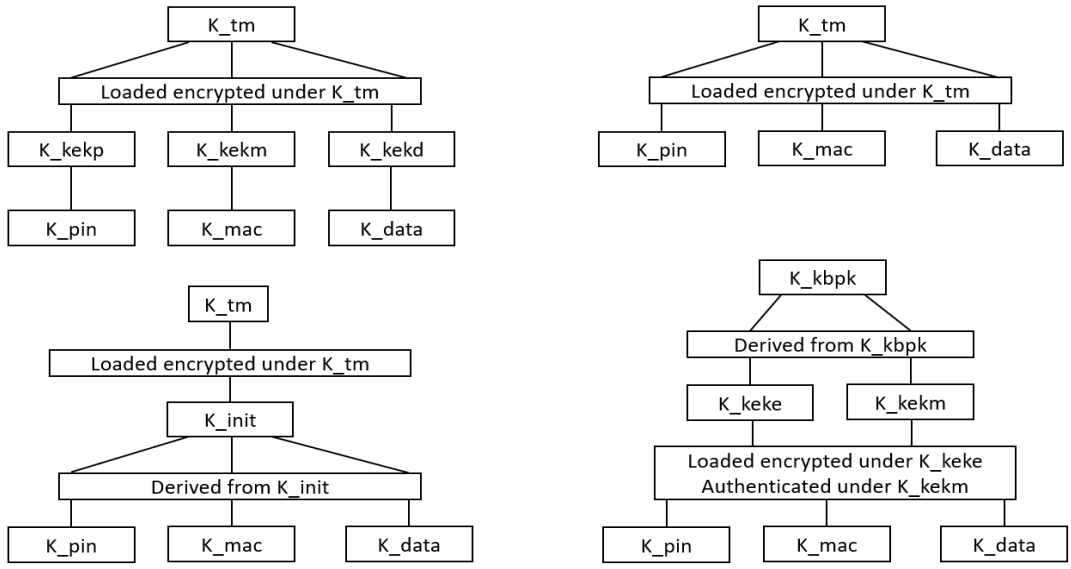

Figure 1 – Example key hierarchies

So, in a simple example we may have the PIN key Kp loaded encrypted under the Master Key Km, and the data key Kd also loaded under the same master key Km. Consider an encryption function E that encrypts plaintext P to ciphertext C, using input key K. This could be denoted as E(K, P) = C. The reverse of this function would be the decryption function D, performing the operation D(K, C) = P.

For PIN key loading we may then have encrypted forms of the PIN key K_p, which are secured under encryption using the master key K_m, as illustrated below.

Loaded PIN Key = E(K_m, K_p)

Loaded data Key = E(K_m, K_d)

In many key management implementations, it is common to have a need to create new keys from an existing key. For example, in Derived Unique systems, new working keys are created for each transaction on the payment acceptance device. Even in Master/Session key management systems, it is common to have a single 'Base Derivation Key' (BDK) which is used to create unique Master keys for each individual payment acceptance device.

ISO11568 considers three broad methods for the creation of a cryptographic key:

1) Non-repeatable key generation
2) Repeatable key generation
3) Key calculation

Non-repeatable key generation involves the use of random or pseudo-random functions, where it is computationally infeasible to generate the same output state. This may be through the use of a true random number generator to directly generate a key with sufficient entropy, or through use of non-repeating values input to a pseudo-random number generator or key agreement scheme.

A repeatable key generation process uses a non-reversible process, such as a pseudo-random number generator (PRNG), with a predictable and known seed (so that process can be repeated). Commonly in payments this would be a process using a One Way Function (OWF) or PRNG seeded by the serial number of the payment acceptance device to generate the device unique master key.

Another method of key creation is 'key calculation', which is a reversible process applied to an existing key to create a new key. In payment systems, this is commonly implemented through a so-called 'variant' process, where values are XOR'd across an existing plaintext key to change the value of that key to 'calculate' the value of a new key (the term 'calculate' is used specifically in ISO11568). Because variants are reversible, it is a requirement that they are not exposed beyond the security boundary in which they are created (e.g. the payment acceptance device in which the original key on which the variants are applied). For this reason, amongst others, the use of variants has been deprecated in many newer key management systems.

Because of the early adoption of encryption within the financial services industry, key management systems have often spanned multiple changes in cryptographic algorithms or key sizes. ANSI X9.24 DUKPT, for example, has three versions – one for single DES, one for Triple DES (TDES), and one for AES. In Master/Session key management systems which implement TDES, it is common for the two 64 bit 'halves' of the DES key to be separately encrypted with the master key, as shown below:

Loaded Key = E(K_m, K_pl)||E(K_m, K_Pr)
Where K_pL = the left most 64 bits of the 128 bit TDES key Kp
K_pR = the right most 64 bits of the 128 bit TDES key Kp
|| is a concatenation operation

The mode of operation used may vary, but ECB is often used in practice.

Attacks on Payment Key Management

It may now start to become clear that there are some potentials for exploiting the ways in which financial key management is performed as outlined above. For example, although the brute force exhaustion of a TDES key remains computationally infeasible, exhaustion of the key domain for a single DES key is very easily achieved with todays computing power. When a TDES working key is loaded using ECB encryption of the two key halves, it becomes possible to exploit this.

If an adversary were to take a single, encrypted TDES key and separate out the two halves, each of these could be duplicated and loaded as their own TDES key – but with the effect that the loaded key would operate as a single DES key equivalent in value to that key half. Using the PIN key example we have from above:

Loaded Key = E(K_m, K_pl)||E(K_m, K_pr) eKm(Kp_L)|eKm(Kp_R)
Adversary Key1 = E(K_m, K_pl)||E(K_m, K_pl)
Adversary Key2 = E(K_m, K_pr)||E(K_m, K_pr)

The adversary intercepts the attempt to load the genuine Loaded Key, and generates from this Adversary Key1 and Adversary Key2. Either one of these keys may be then loaded into the payment acceptance terminal and will properly decrypt into a key which has the same value for each of its two 64 bit halves. Because two key Triple DES operations are performed using an encrypt-decrypt-encrypt process, this results in the key being the equivalent of a single DES key of value equal to the plaintext of the key half used (either K_pl or K_pr).

The attacker only has to obtain a few plaintext/ciphertext pairs to then brute force that key, and then can either load the other key half (and repeat) or load the full key (with both key halves) and brute force the remaining 64 bits they do not know. In either case, the attacker has now successfully obtained the plaintext value of that working key, despite it being loaded encrypted.

This attack works because it is possible to manipulate the encrypted key value without detection. However, in systems where multiple algorithms that support the same master key are implemented, it can be possible to exploit a similar attack without changing the key at all. For example, we may have a system where both TDES and AES PIN keys are supported. If an AES key of 128 bits can be loaded using the function that is intended to load a TDES key, then it may allow for the use of that key using the weaker algorithm.

The notion of parity bits within a TDES key may imply that such an attack would be likely to fail, but it is common in many systems for parity bits to be ignored.

Another attack exploits the fact that there are multiple purposes a key may have in a payment system, and often the operations performed by a key are constrained by that purpose. For example, a PIN key within a payment terminal is only permitted to encrypt PINs – it cannot decrypt a PIN, or perform cryptographic operations on any other data. Similarly, a data key may never be used directly on a customer PIN.

However, an arbitrary payment device which is loading an encrypted key does not inherently know what purpose that key has until it is informed of that purpose. Traditionally, this has been through the command used – “load PIN key” or “load data key” – as the keys themselves have historically not carried any metadata that describe their use. This presents an opportunity for an attacker to alter the purpose of a key during loading.

If we have a system which has two keys, K_p for encrypting PINs and K_d for encrypting (or decrypting) data, and both are loaded into the device encrypted with the master key Km using the commands:

Load_PIN_key(E(K_m, K_p))
Load_data_key(E(K_m, K_d))

An adversary may intercept these commands and alter them so that the PIN key is loaded into both the PIN and data encryption functions, as per below:

Load_PIN_key(E(K_m, K_p))
Load_data_key(E(K_m, K_p))

This now allows for an attacker to use the data decrypt function to decrypt any PIN block output by the device. Even if the system only allows for data encryption, not data decryption, it is trivial for an attacker to use the encrypt function to exhaust the PIN domain (as customer PINs are commonly only four decimal digits in length, with an allowed maximum of 12 digits).

These problems do not only affect payment acceptance terminals. The Hardware Security Modules (HSMs) which are used to store and manage cryptographic keys at the banking

backend systems also often rely on key encryption under some master storage key to protect the working keys they use during operation. Many HSMs do not actually store more than a handful of keys, and all other keys (which may be many hundreds to thousands of unique keys) are managed externally encrypted under a 'master storage key'.

Key Blocks and Key Wrapping

To prevent the possibility of key manipulation attacks on both acceptance and backend systems, standards have been introduced to provide metadata describing the use and type of an encrypted key, as well as providing integrity and authenticity checks over the key and associated metadata. This is known as 'key wrapping', or storing the key in a 'key block'. This concept has been introduced and analyzed previously[v], but for the purposes of this paper the primary implementations referenced are TR-31 and ISO20038.

In these standards, a cryptographic key is conveyed encrypted along with unencrypted metadata that describes the type and purpose of the key. This general format is illustrated below.

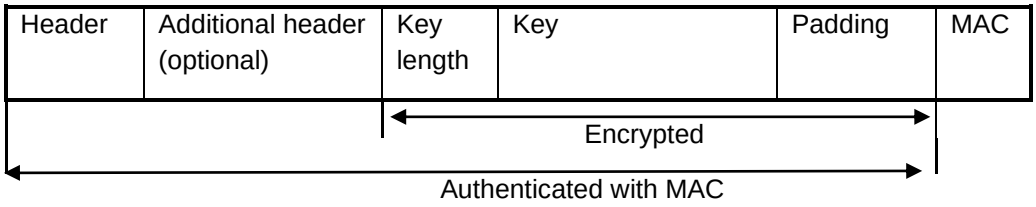

Figure 2 – Illustration of a wrapped key

In line with key management best practice of ensuring a unique key for each unique purpose, key wrapping process involved two different keys, one for use as a KEK and one for the creation of the MAC. The encryption key is used to protect the confidentiality of the key during transport, and the MAC is used to protect the integrity of the key (to prevent key manipulation as described above) as well as the integrity of the key metadata.

In both TR-31 and ISO20038 these keys are created from a base 'Key Block Protection Key' (K_bpk, or KBPK). In both standards there are two ways in which the MAC and KEK keys can be produced, a variant method and a key derivation method, although the variant method is considered deprecated for new use in many standards which reference these. The process used to generate the keys and encrypt/MAC the key block for both the variant and derivation method is illustrated below.

The input data used for the derivation of each key is dependent upon the purpose of the key (MAC or KEK) as well as the mode of operation used, and other optional data which may be supplied.

The security of this key wrapping process is a function of the security of the cryptographic algorithm(s) used in the generation of the encrypted key block and the MAC, which may be AES or TDES. The MAC function used is CMAC (as defined in NIST SP800-38B), and the mode of operation used for the encryption of the key may be either CBC or CTR mode, depending on the configuration used (which is then defined in the header and key derivation process).

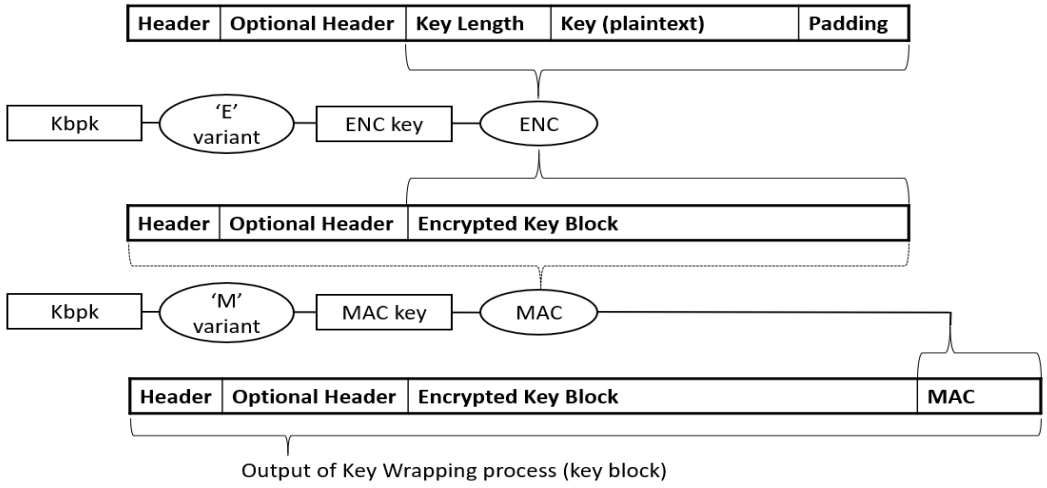

Figure 3 – The derivation and use of encryption and authentication keys TR-31 'A' and 'C' versions

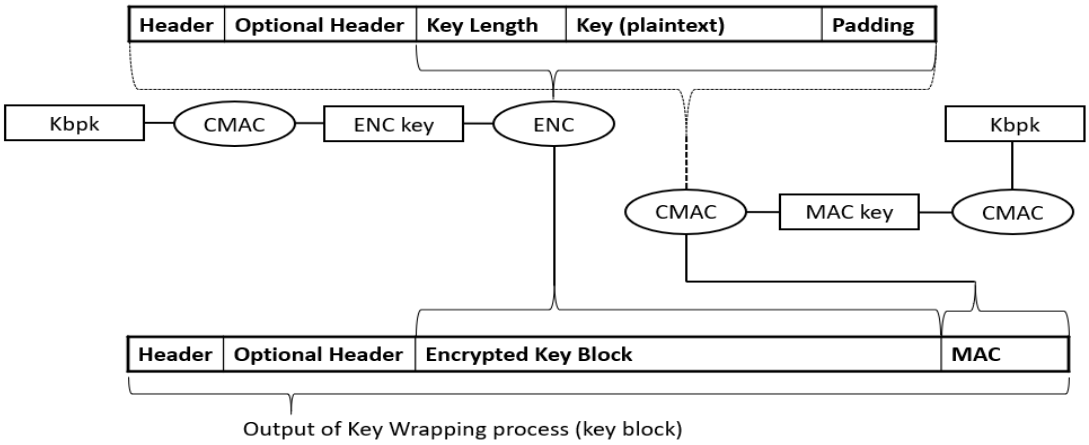

Figure 4 – The derivation and use of encryption and authentication keys in ISO20038 / TR-31 'B' and 'D' versions

It can be noted that the two types of TR-31 (the 'A' and 'C' variant versions, and the 'B' and 'D' CMAC versions) differ in how they approach the way in which the MAC is calculated. In the variant versions of TR-31, a CBC MAC is calculated across the key block once it has been constructed with the key header and encrypted key block. This 'encrypt then MAC' approach provides value in allowing for any manipulated key block to be rejected prior to any further parsing of the key block, or use of the actual KEK that protects the key.

However the more recent version of TR-31, which uses CMAC operations to derive the working KEK and authentication keys for the key block, calculates the MAC across the key block prior to the encryption of the key. This means that in all cases, the key block must be (at least partially) parsed and unencrypted prior to determining even if it is has been subject to manipulation. This exposes additional code surface within the key usage areas to attack, as well as exposing the KEK to side channel and other types of attack as it is used.

However, for the purposes of comparing IBM key block implementations to TR-31, it will be assumed that a MAC-then-encrypt approach is acceptable as used in TR-31 'B' and 'D' versions.

IBM WRAPENH3 Key Wrap Implementation

IBM have implemented a proprietary key wrapping method designated WRAPENH3 (a shortening of wrap enhanced 3). Details on this were supplied through various documents and discussions with IBM[vivii]. This system is an increment of existing IBM wrap modes that utilize a 'control vector' (CV) to provide metadata on the key, but instead of relying upon a binary addition of this CV onto the KEK, the metadata is bound to the wrapped key using a CMAC. This document does not detail the implementation of the existing WRAPENH1 and WRAPENH2 modes used in IBM systems, and concerns itself solely with the new WRAPENH3 format.

In this format, the key metadata and key itself are bundled into a 64 byte block that forms the key bundle / wrap. This block is defined in Table 1 – IBM WRAPENH3 format on the next page (note this contains details on settings for key wrapping other than WRAPENH3). This block includes details on the length and use of the key, as well as version details for the wrapping process itself, the encrypted key value and the CMAC output, amongst other data.

As with other key wrapping standards, the keys used for encryption and authentication are derived separately from a single master key block protection key. This process utilizes a SHA256 HMAC operating as a Key Derivation Function (KDF) in counter mode, as per NIST SP800-108[viii]. Specifically, this is:

K = HMAC(Kbpk, 0x0000||Label||0x00||[Length])
Where 'Length' = Four byte big-endian bit length of the key being derived
'Label' = The ASCII characters WRAPENH3KEY-ENCR for the KEK
= The ASCII characters WRAPENH3KEY-CMAC for the authentication key
= The ASCII characters PCI-HSM_ENC_2020 for KEKs used in PCI mode
= The ASCII characters PCI-HSM_MAC_2020 for auth keys in PCI mode

For WRAPENH3, the key wrapping process itself is defined as follows:

1) Set the values of the Control Vector 1 (CV1) as required for the key type, version, etc.
2) Set the value for Control Vector 2 as all zeros.
3) Generate the authentication and key encrypting keys for the key wrapping process.
4) Place the plaintext values of the key into the key block. The length of the key is represented by the first key that has all zero values, so a single length key has the key value stored in the K1 position, with K2 and K3 loaded with zeros. A double length key has the first half of the key loaded in K1, the second half loaded in K2, and all zeros loaded into K3. A triple length key has all key positions loaded with their key values.
5) Calculate a CMAC across the entire block, using the derived authentication key. Place the output of this CMAC calculation into the location for Control Vector 2.
6) Chain the parts of the plaintext key together using SHA256, in the following process:
    a. K3C = K3
    b. K2C = K2^Trunc_SHA256(K3)
    c. K1C = K1^Trunc_SHA256(K2)
    d. KC = K1C||K2C||K3C
    Where: Trunc_SHA256 is the first 64 bits of the SHA256 of that part of the key
    ^ is the XOR operator
    || is concatenation of two values
7) Encrypt the key using the derived key encryption key across the chained key parts (KC) using the CBC mode of operation.
8) Place the encrypted key parts into the respective K1, K2, and K3 slots in the key block.

The key block format is defined in the table below.

Offset (bytes)	Length (bytes)	Field	Definition
0	1	Token flag	EMPTY_TOKEN_FLAG 0x00 INTERNAL_TOKEN_FLAG 0x01 EXTERNAL_TOKEN_FLAG 0x02 (all other values reserved)
1	1	RFU	RFU
2	2	Oldmkvp	Legacy field, now always loaded with 0x0000
4	1	Version	0x00 (for WRAPENH3 version)
5	1	RFU	RFU
6	1	Flag byte 1	MASK_KEY 0x80 // encrypted key & MKVP present MASK_CV 0x40 // CV in token has been applied S390-ONLY :: MASK_NOCV 0x20 // KEK used for NOCV processing MASK_AKEK 0x10 // ANSI KEK (AKEK) MASK_AKEK_DOUBLE 0x08 // AKEK is double-length key MASK_AKEK_PART_NOTARIZED 0x04 // AKEK is partially notarized MASK_ANSI_PARTIAL_KEY 0x02 // key is an ANSI partial key MASK_XPORT_PROHIB 0x01 // prohibit export when bit is 0b1
7	1	Flag byte 2	bit 0-2: wrap method: '000xxxxx' WRAP_TK_LEGACY = 0, ECB/legacy method '001xxxxx' WRAP_TK_ENH_CBC = 1, Chain Keys with SHA-1, SHA-256 KDF->KEK, variant KEK, CBC encryption '010xxxxx' WRAP_TK_ENH_2 = 2, Chain Keys with SHA-256, SHA-256 KDF->KEK, variant KEK, CBC encryption '011xxxxx' WRAP_TK_ENH_3 = 3, Chain Keys with SHA-256, SHA-256 KDF-> MAC key, TDES-CMAC, SHA-256 KDF->KEK, CBC encryption bit 3-5: 'xxx000xx' reserved bit 6: 'xxxxxx1x' used for legacy case pre-2009 code was not clear on how this is used, but it is checked bit 7: 'xxxxxxx0' reserved
8	8	Mkvp	KEK or Master Key verification pattern
16	8	Left key part (K1)	First 8 bytes of the chained and encrypted (T)DES key
24	8	Right key part (K2)	Second 8 bytes of the chained and encrypted (T)DES key
32	8	Control Vector 1 (CV1)	The value of the Control Vector 1
40	8	Control Vector 2 (CV2)	The value of the Control Vector 2, used for CMAC in WRAPENH3
48	8	Third key part (K3)	The third 8 bytes of an encrypted 168 bit (T)DES key, or random data
56	3	RFU	RFU
59	1	Token Marks	TM_KEY_LENGTH_SINGLE 0x00 TM_KEY_LENGTH_DOUBLE 0x10 TM_KEY_LENGTH_TRIPLE 0x20 TM_CDMF_DATA 0x80 TM_CDMF_KEK 0x80 TM_DES_DATA 0x00 TM_SYSTEM_DEFAULT_KEK 0x00 TM_DES_KEK 0x40 TM_KEY_LENGTH_MASK 0x30 TM_TOKEN_MARKS_MASK 0xC0 TM_TOKEN_MARKS_SINGLE_MASK 0xCF TM_RESERVED 0x0F
60	4	Token Validation Value (TVV)	Legacy checksum

Table 1 – IBM WRAPENH3 format

The attributes of the key are defined in the Control Vector 1 value, which are provided in detail in Appendix I to this document. These provide for specific values to be used for defining the key type in the first byte, with variations on that specific key type defined in the third byte of the control vector. The fifth byte defines what subordinate keys a key that is defined as a KEK may operate on, and the sixth and eighth bytes are used for general key metadata such as exportability, etc.

Some of the general metadata values – such as wrap format and key length – are not used in the WRAPENH3 format, and only exist for legacy reasons.

The overall Control Vector format is summarized below. For specific details on all values, refer to the Appendix.

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Key Type		Key type specifics		Permitted subordinate keys for KEKs	General metadata		General metadata

Table 2 – High level IBM control vector format

Examples of default / common control vector values are provided in the table below (copied from reference [vi]

Type	**Control Vector**	**Description**
Key encrypting keys		
EXPORTER	00 41 7D 00 03 41 00 A0 00 41 7D 00 03 21 00 A0	Used to encrypt a key taken from this local node CCA 6.2+: allowed to be triple-length key type.
IKEYXLAT	00 42 42 00 03 41 00 A0 00 42 42 00 03 21 00 A0	Used to decrypt an input key token in a key translation service that decrypts an external input key token under an IKEYXLAT KEK, then encrypts the key material as a new external output key token under an OKEYXLAT KEK.
IMPORTER	00 42 7D 00 03 41 00 A0 00 42 7D 00 03 21 00 A0	Used to decrypt a key brought to this local node. CCA 6.2+: allowed to be triple-length key type.
OKEYXLAT	00 41 42 00 03 41 00 A0 00 41 42 00 03 21 00 A0	Used to encrypt an output key in a key translation service that decrypts an external input key token under an IKEYXLAT KEK, then encrypts the key material as a new external output key token under an OKEYXLAT KEK.
Data protection keys		
CIPHERXI	00 0C 50 00 03 C0 00 A0 00 0C 50 00 03 A0 00 A0	Used to decrypt ciphertext during text translation from 1 cipher key to another cipher key.
CIPHERXO	00 0C 60 00 03 C0 00 A0 00 0C 60 00 03 A0 00 A0	Used to encrypt ciphertext during text translation from 1 cipher key to another cipher key.
CIPHERXL	00 0C 71 00 03 C0 00 A0 00 0C 71 00 03 A0 00 A0	Used to decrypt or encrypt ciphertext during text translation.

Type	Control Vector	Description
CIPHER	00 03 71 00 03 41 00 A0 00 03 71 00 03 21 00 A0	Used only to encrypt or decrypt data. CCA 6.2+: allowed to be triple-length key type.
DATAC	00 00 71 00 03 41 00 A0 00 00 71 00 03 21 00 A0	Used only to encrypt or decrypt data.
DECIPHER	00 03 50 00 03 41 00 A0 00 03 50 00 03 21 00 A0	Used only to decrypt data. CCA 6.2+: allowed to be triple-length key type.
ENCIPHER	00 03 60 00 03 41 00 A0 00 03 60 00 03 21 00 A0	Used only to encrypt data. CCA 6.2+: allowed to be triple-length key type.
Data integrity keys		
DATAM	00 00 4D 00 03 41 00 A0 00 00 4D 00 03 21 00 A0	Used to generate or verify a MAC.
DATAMV	00 00 44 00 03 41 00 A0 00 00 44 00 03 21 00 A0	Used to verify a MAC code; cannot be used in MAC-generation
MAC	00 05 4D 00 03 41 00 A0 00 05 4D 00 03 21 00 A0	Used to generate or verify a MAC. CCA 6.2+: allowed to be triple-length key type.
MACVER	00 05 44 00 03 41 00 A0 00 05 44 00 03 21 00 A0	Used to verify a MAC code; cannot be used in MAC-generation CCA 6.2+: allowed to be triple-length key type.
PIN-processing keys		
IPINENC	00 21 5F 00 03 41 00 A0 00 21 5F 00 03 21 00 A0	Inbound PIN encrypting key, used to decrypt a PIN block CCA 6.2+: allowed to be triple-length key type.
OPINENC	00 24 77 00 03 41 00 A0 00 24 77 00 03 21 00 A0	Outbound PIN encrypting key, used to encrypt a PIN block CCA 6.2+: allowed to be triple-length key type.
PINGEN	00 22 7E 00 03 41 00 A0 00 22 7E 00 03 21 00 A0	Used to generate and verify PIN values CCA 6.2+: allowed to be triple-length key type.
PINVER	00 22 42 00 03 41 00 A0 00 22 42 00 03 21 00 A0	Used to verify, but not generate, PIN values CCA 6.2+: allowed to be triple-length key type.
Key-generating keys		
DKYGENKY	00 71 44 00 03 41 00 A0 00 71 44 00 03 21 00 A0	Used to generate a diversified key based on a key-generating key.

Alteration of any of the control vector bits, or of the key itself, will result in changing the input data used to generate the CMAC, and therefore invalidating the CMAC when it is checked. Brute forcing of the potential values of the CMAC to attempt to load a modified block would require an attack across the 64 bit domain of the CMAC to brute force a specific value.

IBM Variable Length Symmetric Key Block

IBM have also implemented another key wrapping method, designated as a Variable Length Symmetric Key block. This system is based on the key wrapping standard published by ANSI in X9.102[ix], and utilizes AES based protection keys for the purposes of wrapping another key for storage or distribution. For the purposes of this report, we refer to this key block under review as the VLS key block.

Unlike TR-31 or ISO20038, which use separate keys for confidentiality and integrity, the VLS key block uses a single key to encrypt a body of data that contains the key as well as specific key metadata, using a mode of operation designated as AESKW. This mode of operation is specified in ANSI X9.102, and also defined in NIST SP800-38F[x], specifically for the purposes of key wrapping.

In their 2007 paper "Deterministic Authenticated-Encryption"[xi], Rogerway et al note that no formal proof for the security of this mode exists, but comment that "… we find it likely the mechanism is correct". The use of ANSI X9.102 and AESKW is additionally referenced in the PCI SSC document "Information Supplement: Cryptographic Key Blocks"[xii], within the section defining the meaning and purpose of key wrapping, implicitly applying approval to this method of key wrapping.

The PCI paper also notes, however, that there are many different ways to implement key wrapping and ANSI X9.102 leaves open specifics of the implementation. This document discusses the specifics of the IBM implementation utilizing AES keys for the key wrapping process.

The key wrapping method used in VLS is performed in two stages. The first stage takes the key metadata (as defined in Appendix I) and hashes this data using SHA256. This hash is then taken, along with some static details on the type and length of the hash used for this type of key block, and placed into a subsequent block – along with the plaintext key to be wrapped, an ICV, and padding – that is encrypted with the wrapping key using the AESKW mode of operation.

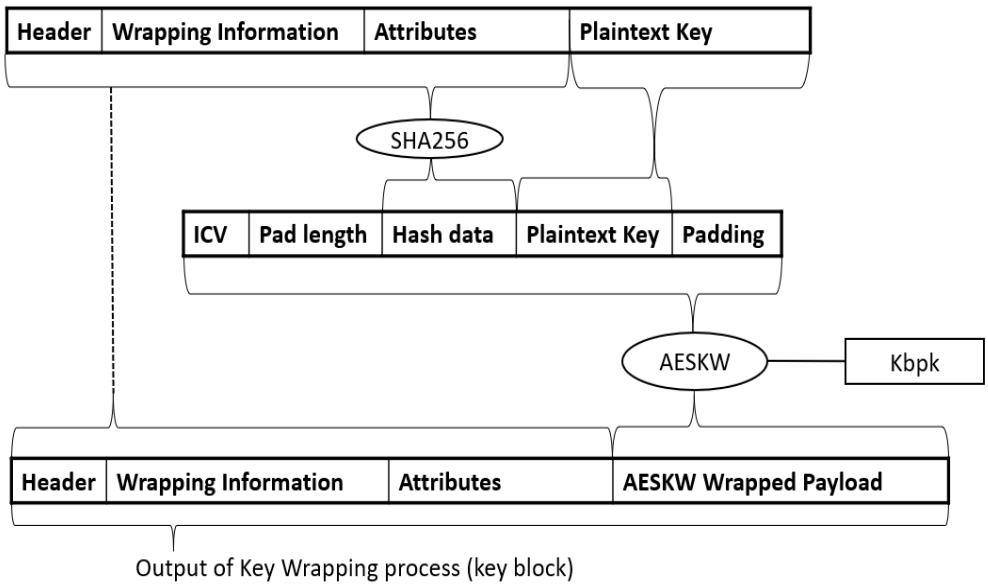

Figure 5 – Illustration of IBM VLS key wrapping

This process is illustrated above, and defined formally below.

Wrapped Key =
(AssociatedData)||AESKW(K_aeskw, [ICV||PadLength||HashData||Key||Padding])

Where Associated Data = The key block metadata, including header, wrapping information, and key attributes, as defined in Table 3

K_aeskw = The key to be used as the wrapping key

ICV = A 6 byte constant 0xA6A6A6A6A6A6
This is the default ICV value defined in NIST SP800-38F for AESKW

PadLength = The length of the padding applied, set so that the length of the key to be encrypted plus the padding is always the maximum key length for that algorithm XXX Need to reference v1 payload only XXX

HashData = The length, type and output of the hash function across the associated data block. Length and type are fixed for this key wrapping type to SHA256

Key = The key to be wrapped

Padding = The number of 0x00 bytes as defined by the padding length.

In this method, the key properties are set in the associated data block and the inclusion of a hash over this data into the encrypted block is used as the method to bind this key data to the key itself.

The value of the key properties contained in the Associated Data block are provided in the next page in Table 3. This defines the key wrapping method used, the version, the length and purpose of the key, etc. Changing any value within the Associated Data block would cause the SHA256 calculation across this block to produce a different value, therefore failing the check when they key is decrypted.

This key wrapping process does not utilize keys separately derived by the operating device (HSM, Point of Interaction etc) before the wrapping / unwrapping process, as do both TR-31 and the IBM WRAPENH3 key wrapping methods. Therefore this mode also requires the use of the KEK before authentication of the key block can be performed.

Offset (bytes)	Length (bytes)	Field	Definition
Header			
0	1	Token flag	INTERNAL_TOKEN_FLAG 0x01
1	1	RFU	RFU
2	2	Length	Length in bytes of total associated data block
4	1	Version	0x05 (for VLS version)
5	3	RFU	RFU
Wrapping Information			
8	1	Key material state	0x00 No key is present 0x01 Key is clear (this would be non-compliant to PCI requirements) 0x02 Key is wrapped with a transport key 0x03 Key is wrapped with the AES master key
9	1	Key verification pattern (KVP)	0x00 No KVP, key is not present, plaintext, or RSA wrapped 0x01 AESMK (8 leftmost bytes of SHA256) 0x02 KEK (8 leftmost bytes of SHA256)
10	16	KVP value	0x00 The key material state is no key is present 0x01 The key material state is the key is clear 0x02 The key material state is the key is wrapped with a transport key 0x03 The key material state is the key wrapped with the AES master key
26	1	Encrypted key wrapping method	0x00 No key wrapping method 0x02 AESKW 0x03 PKOAEP2, RSAES-OAEP RSA PKCS#1 v2.1
27	1	Hash algorithm used	0x00 No hash 0x01 SHA-1 0x02 SHA-256 0x03 SHA-384 0x08 SHA-512
28	1	Payload format version	0x00 v0 payload 0x01 v1 payload Wrapping methods uses fixed key length
29	1	RFU	
Attribute Data			
30	1	Attribute data section	0x01 Version 1
31	1	RFU	
32	2	Attribute data length	Length in bytes of all attribute data
34	1	Optional key label length	Length in bytes of the optional key label
35	1	IBM optional data length	Length in bytes of optional IBM extended attribute data
36	1	User definable data length	Length in bytes of optional user definable attribute data
37	1	RFU	
38	2	Key length	Length in bits of key or wrapped key, including padding
40	2	RFU	
41	1	Algorithm type	The algorithm for which the wrapped key can be used: 0x01 = (T)DES, 0x02 = AES, 0x03 = HMAC
42	2	Key type	The purpose of the wrapped key 0x0001 Cipher 0x0002 MAC 0x0003 KEK export 0x0004 KEK import 0x0005 PIN protection 0x0006 PIN calculation 0x0007 PIN reference value 0x0008 DESUSECV 0x0009 DUKPT key generation key 0x000A AES and (T)DES secure messaging, such as for EMV 0x000B Key generation key
44	1	Key field usage count (Kuf)	The number of key usage fields contained below
45 -	Variable	Key usage fields	

Table 3 – VLS key block structure (excluding AESKW encrypted key)

PCI SSC Key Wrapping Mandates

One of the entities that governs the standards used for maintaining security across the card payment landscape is the Payment Card Industry Security Standards Council (PCI SSC). This entity has published requirements for the use of key wrapping in systems that fall under the purview of their standards, with the primary resources being the PCI PIN standard[xiii] (which covers the key management requirements for entities that process PIN based transactions), and two information supplements' published in 2017[xiv] and 2019[xv].

In these documents, requirements for the implementation and use of key wrapping methods for cryptographic keys which provide security for PIN based transactions are outlined. This document does not attempt to cover the specific dates of these mandates, but instead outline the actual requirements themselves so that conclusions can be drawn concerning the compliance of IBMs key wrapping implementations.

In both of the information supplements, the Executive Summary indicates acceptable methods for implementing the requirements of key wrapping to include:

- A MAC computed over the concatenation of the clear-text attributes and the enciphered portion of the Key Block, which includes the key itself,
- A digital signature computed over that same data,
- An integrity check that is an implicit part of the key-encryption process such as that which is used in the AES key-wrap process specified in ANSI X9.102.

In the 2019 supplement, several Frequently Asked Questions – with associated answers – are provided, including the clarification that key wrapping methods do not need to provide for 'directionality' of keys, i.e. a key wrapping method does not need to be able to differentiate between 'send' keys, which perform encryption operations on some data, as opposed to 'receive' keys, which would decrypt such data (this is clarified in Q14 of that document).

An update to the PCI PIN requirements for key wrapping was provided in Jan of 2021, with the following FAQ:

> ***Q 34 January 2021****: Encrypted symmetric keys are required to be managed in structures called key blocks. These key blocks may be as defined in ANSI or ISO standards, or equivalent proprietary methods. PCI recently published detailed HSM and POI FAQs defining the characteristics that an 'equivalent' key block must possess, and the process for assessment. How does this impact PIN Security assessments?*
> ***A*** *Until January 2023, Service Providers can continue to operate using existing proprietary methods that have not yet been validated under the defined key block equivalency process. Any newly developed proprietary methods must undergo the defined key block equivalency process prior to any implementations. After December 2022, proprietary methods not validated as equivalent will be considered non-compliant.*

Examples of Key Wrapping Implementations

In this section we review some examples of both TR-31 and IBM proprietary key blocks, to demonstrate how they function practically and highlight similarities between the implementations.

TR-31 Example Key Blocks

Two TR-31 example key blocks are shown below:

```
A0136V0TN00S0200102CIBMC012400227E000341000000227E0003210000PB047F5787
857B413A01A880461CB19203B0F2D9E3E5326133B9D29036D35BEC873C95F22E81
```

```
B0144P0TE00S0200102CIBMC0124002477000341000000247700032100 00PB04C71F19
9CC5A13FECEAAF94EC3CC4C3025787E709BC8101236F51736F93421D65CABAD5E97A7F
D11B
```

By simply viewing the first digit of each of these key blocks, it is possible to determine that one is definitely not compliant to the requirements – that is because the first digit indicates the key block version, and only versions ‘B’ and ‘D’ are compliant to PCI PIN key block rules. In the first of the examples above, the version is ‘A’, indicating that it uses a variant method of diversifying the Key Block Protection Key into the encryption and authentication keys. This violates the PCI PIN requirements.

Examining the first example block in more detail, we have the following:

Byte Number	TR-31 assignment	Value in example	Description for this key block
0	Version ID	A	Key block type 'A', using variants to calculate working keys
1-4	Key block length	0136	Total key block length is 136 characters
5-6	Key usage	V0	Key in block is a PIN verification key
7	Algorithm	T	The wrapped key is a TDES key
8	Mode of use	N	No restrictions on mode of use
9-10	Key version number	00	No key versioning is implemented for this key block
11	Exportability	S	The key is sensitive and can only be exported under a KEK
12-13	Number of optional blocks	02	The key block contains two optional header blocks
14-15	RFU	00	Reserved
16-17	Optional Block ID	10	The ID of the first optional block (is 10). A numeric value indicates a proprietary optional block.
18-19	Optional block length	2C	This optional block has a length of 2C, or 44 characters (expressed in decimal)
20-59	Optional block content	IBMC0124 - IBM proprietary block carrying DES control vectors 00227E0003410000 – Control vector 1 00227E0003210000 – Control vector 2	
60-61	Optional Block ID	PB	This is the ID for a padding block, used to space the key block out to relevant block lengths as required
62-63	Optional block length	04	The length of this optional block is 04, which means no further data is included
64-128	Encrypted key	7F5787857B413A01A880461CB19203B0F2D9E3E5326133B9D29036D35BEC873C This is the wrapped key, length indicator, and associated padding all encrypted with the KEK varianted from the KBPK	
129-135	MAC	95F22E81 The first four bytes of a CBC MAC used to authenticate the key block, calculated over the previous values (including encrypted key) using the auth key varianted from the KBPK	

Table 4 – TR-31 'A' type, variant based key block example

For the second key block, the decomposition is:

Byte Number	TR-31 assignment	Value in example	Description for this key block
0	Version ID	B	Key block type 'B', using CMAC to derive the working keys
1-4	Key block length	0144	Total key block length is 144 characters
5-6	Key usage	P0	Key in block is a PIN encryption key
7	Algorithm	T	The wrapped key is a TDES key
8	Mode of use	E	The key can be used for encryption or wrapping only
9-10	Key version number	00	No key versioning is implemented for this key block
11	Exportability	S	The key is sensitive and can only be exported under a KEK
12-13	Number of optional blocks	02	The key block contains two optional header blocks
14-15	RFU	00	Reserved
16-17	Optional Block ID	10	The ID of the first optional block (is 10)
18-19	Optional block length	2C	This optional block has a length of 2C, or 44 characters (expressed in decimal)
20-59	Optional block content	IBMC012400227E000341000000227E0003210000	
60-61	Optional Block ID	PB	This is the ID for a padding block, used to space the key block out to relevant block lengths as required
62-63	Optional block length	04	The length of this optional block is 04, which means no further data is included
64-128	Encrypted key	C71F199CC5A13FECEAAF94EC3CC4C3025787E709BC8101236F51736F93421D65 This is the wrapped key, length indicator, and associated padding all encrypted with the KEK CMAC derived from the KBPK	
129-143	MAC	CABAD5E97A7FD11B The first four bytes of a CBC MAC used to authenticate the key block, calculated over the previous values (including encrypted key) using the auth key CMAC derived from the KBPK	

Table 5 – TR-31 'B' type, CMAC derivation based key block example

IBM Sample Key Blocks

Examples of the IBM WRAPENH3 and VLS key blocks were also supplied, as shown below:

WRAPENH3 sample key block:

```
020000000000C0600000000000000000E0DCEFE482282605116F7A4CC3652AFD000371
0003600081AE1F4C7FD672C0E83C62B185E7411B890000000008692362
```

VLS sample key block:

```
[0100008c05000000030149da4dd4e8781573000000000000000020201000100]
[001e0000000028000020003O4fc000000e000f80003e00000000505bfb9d631]
[8227f586edf221d05d41f908aae3ea49ede64347451556dad13030db164ba956]
[82664f496a5c85b6ba34c3202bd5491552ba23ede40850bd5f32b5a717dba2e3]
[74d24f5aee60f3122c10a265]
```

A decomposition of the WRAPENH3 key block is tabulated below:

Byte Number	**WRAPENH3 Assignment**	**Value in example**	**Description for this key block**
0	Token type	02	External token
1	RFU	00	
2-3	Oldmkvp	0000	
4	Version	00	This must be 00 for WRAPENH3
5	Reserved	00	
6	Flag byte 1	C0	
7	Flag byte 2	60	011xxxxx = WRAPENH3
8-15	Mkvp	0000000000000000	
16-23	Left key part (K1)	E0DCEFE482282605	Encrypted K1
24-31	Right key part (K2)	116F7A4CC3652AFD	Encrypted K2
32-39	Control Vector 1 (CV1)	0003710003600081	Data encryption key
40-47	Control Vector 2 (CV2)	AE1F4C7FD672C0E8	CMAC output across key block
48-55	Third key part (K3)	3C62B185E7411B89	Encrypted K3 or random data
56-58	RFU	000000	
59	Token Marks	00	
60-63	Token Validation Value	08692362	

Table 6 – IBM proprietary WRAPENH3 key block example

Decomposition of the VLS key block is performed in the table below:

Byte Number	VLS Assignment	Value in example	Description for this key block
0	Token flag	01	
1	RFU	00	
2-3	Length	008c	Length of key block is 8c, or 140 characters
4	Version	05	VLS version
5-7	RFU	000000	
8	Key material state	03	Key is wrapped with an AES master key
9	Key verification pattern (KVP)	01	Key verification pattern is AESMK
10-25	KVP value	49da4dd4e8781573000000000000000	
26	Encrypted key wrapping method	02	AESKW key wrapping is used
27	Hash algorithm used	02	SHA256 is used as the hash algorithm
28	Payload format version	01	V1 payload
29	RFU	00	
30	Attribute data section	01	
31	RFU	00	
32-33	Attribute data length	001e	Length of attribute data section, from byte 30, is 1E, or 30 characters in decimal
34	Optional key label length	00	
35	IBM optional data length	00	
36	User definable data length	00	
37	RFU	00	
38-39	Key length	0280	Payload length in bits
40	RFU	00	
41	Algorithm type	02	Wrapped key is an AES key
42-43	Key type	0003	Wrapped key is a KEK for key export
44	Key field usage count (Kuf)	04	
45-59	Key usage fields	fc000000e000f80003e00000000505	
60-139	Encrypted key block	bfb9d6318227f586edf221d05d41f908aae3ea49ede64347451556dad13030db164ba95682664f496a5c85b6ba34c3202bd5491552ba23ede40850bd5f32b5a717dba2e374d24f5aee60f3122c10a265	

Table 7 – IBM proprietary VLS key block example

Comparing IBM key blocks to TR-31

From the previous sections, it can be seen that all the described key blocks follow the broad format of [header][encrypted key data][authentication block]. The organization and content of the various parts are different for each key block, as are the derivation and use of the keys used for the encryption and authentication functions. Although TR-31 format 'A' key blocks are deprecated for new use, the use of variants to calculate the working keys continues to be used in the 'C' version of TR-31 based key blocks, as does the use of CBC MAC as the authentication function. Additionally, there is no consistency in TR-31 with regards to encrypt-then-MAC, or MAC-then-encrypt, so it must be assumed for the purposes of comparison that either is acceptable (although encrypt-then-MAC provides many benefits).

The IBM WRAPENH3 format uses the same CMAC operation for authentication across the key and header data as do the 'B' and 'D' TR-31 key block versions, including taking a MAC-then-encrypt approach, thereby providing equivalent protection against modification to the TR-31 versions. The header content provides for a similar range of key attributes in regards to key use, but does lack the key versioning TR-31 provides, which can help protect against injection of expired keys.

The IBM VLS key block uses the AESKW mode of operation, as defined in NIST SP800-38F, rather than a CMAC for authentication. To the best of our knowledge, no formal proof of AESKW exists, but importantly the authenticity of the associated data is provided through inclusion of the SHA256 hash within the AESKW rather than the AESKW algorithm itself. Modification of the key block associated data / header block will force a different SHA256 output when this is calculated on the receiving end, allowing the modification to be detected. Modification of the encrypted portion of the key block is unable to provide predictable changes to the content and will lead to failure of the AESKW authentication process.

IBM WRAPENH3 PCI PIN Compliance

The PCI SSC have outlined[xvi] eight specific requirements that must be met for any key block that is not implemented exactly as per TR-31 or ISO20038. These points are detailed below.

a) It must prevent the loading of PIN, MAC, and/or Data keys - or any keys used to manage these within the key hierarchy - from being used for another purpose. IPEK, KEKs, and derivation keys must be uniquely identified where supported.
b) It must prevent the determination of key length for variable length keys.
c) It must ensure that the key can only be used for a specific algorithm (such as TDES or AES, but not both).
d) It must ensure a modified key or key block can be rejected prior to use, regardless of the utility of the key after modification. Modification includes changing any bits of the key, as well as the reordering or manipulation of individual single DES keys within a TDES key block.
e) Where different key block formats are supported, with some providing the above protections and some not, it must be humanly readable from the key block prior to loading/use which format is implemented. E.g., by looking at the commands sent to the device.
f) It must support all symmetric algorithms implemented by the device(s) that are to use the key blocks.
g) Where asymmetric algorithms are supported, the algorithm type, padding and signature formats must be identified in the key block.
h) It must use NIST approved modes of operation, with separate keys used for confidentially and authenticity. Any keys used must not be related in a reversible way.

In reference to WRAPENH3 for these set of requirements, we note (using the same numbering system as above):

a) The WRAPENH3 format provides key identification, including unique identification of PIN, MAC, Data, KEKs, and Derivation keys in the Control Vector value (as detailed in Table 1 and Appendix I). IPEKs are designated as a specific type of derivation key, which can only be used with ANSI X9.24 DUKPT based derivation operations.
b) The WRAPENH3 format always includes values in the three slots allocated to the key parts. Determination of which of these are valid for key use requires knowledge of the decrypted key parts which provide key length by setting unused key parts to all zeros (so K2 and K3 = all zeros for a single length key, and K3 = all zeros for a double length key). Without knowledge of the plaintext key, keys of different lengths are indistinguishable.
c) The WRAPENH3 process is only defined for use with TDES keys.
d) The WRAPENH3 process implements a CMAC across the length of the block, including the plaintext key values. Modification or manipulation of any of the values in the key block will result in the failure of the CMAC validation, and subsequent rejection of the key block.
e) There are multiple key blocks possible in this format, and the WRAPENH3 format is able to be uniquely identified by the value of 011 in bits 0-2 of the Flag byte 2 value.
f) The WRAPENH3 key block is designed solely for TDES use. AES keys are supported by another key block format by this HSM, which is considered acceptable.
g) Asymmetric algorithms are not supported by this key block format.
h) The key encryption is implemented using CBC, which is a NIST approved mode of operation.

Therefore, the IBM WRAPENH3 algorithms can be seen as entirely equivalent to the key wrapping processes defined in TR-31 and ISO20038, in both security and equivalency criteria

outlined by the PCI SSC. Each use a CMAC process for the generation of an authentication block across the key block, and each use the same algorithms and modes of operation for the encryption of the key itself (and any associated padding).

The differences between the IBM WRAPENH3 and TR-31 / ISO20038 can be best summarized as:

i. A different format is used for the metadata and key storage in the block.
ii. The WRAPENH3 process binds the key parts together with SHA256 prior to encryption (which the TR-31 / ISO20038 process does not).
iii. The WRAPENH3 process derives the KEK and the authentication key using a HMAC based key derivation function, rather than a CMAC based key derivation function used in TR-31 / ISO20038.
iv. WRAPENH3 supports TDES keys encrypted with CBC mode of operation only. AES keys are managed using another key block format in IBM HSM devices.

IBM VLS Key Wrapping PCI PIN Compliance

The PCI SSC have outlined[xvii] eight specific requirements that must be met for any key block that is not implemented exactly as per TR-31 or ISO20038. These points are detailed below.

a) It must prevent the loading of PIN, MAC, and/or Data keys - or any keys used to manage these within the key hierarchy - from being used for another purpose. IPEK, KEKs, and derivation keys must be uniquely identified where supported.
b) It must prevent the determination of key length for variable length keys.
c) It must ensure that the key can only be used for a specific algorithm (such as TDES or AES, but not both).
d) It must ensure a modified key or key block can be rejected prior to use, regardless of the utility of the key after modification. Modification includes changing any bits of the key, as well as the reordering or manipulation of individual single DES keys within a TDES key block.
e) Where different key block formats are supported, with some providing the above protections and some not, it must be humanly readable from the key block prior to loading/use which format is implemented. E.g., by looking at the commands sent to the device.
f) It must support all symmetric algorithms implemented by the device(s) that are to use the key blocks.
g) Where asymmetric algorithms are supported, the algorithm type, padding and signature formats must be identified in the key block.
h) It must use NIST approved modes of operation, with separate keys used for confidentially and authenticity. Any keys used must not be related in a reversible way.

In reference to VLS Key Wrapping for these set of requirements, we note (using the same numbering system as above):

a) The associated data block bound into the key wrapping through the SHA256 function includes designations for the key as a MAC, KEK, data, and key derivation key. Different types of PIN key are also supported, depending on the function (e.g. PIN calculation or PIN encryption)
b) The key block can support a mode where the wrapped key is padded to the maximum key length for that algorithm. This is similar to how TR-31 and ISO20038 function.
c) The algorithm that the key can be used for is defined in the associated data block, and maybe 0x01 to 0x03 for compliant usage, which defines a key to be used for the (T)DES, HMAC, or AES algorithms respectively.
d) Modification of the key or associated data is prevented by the use of the SHA256 across the associated data block, and the use of the AESKW authenticated encryption mode of operation. Changing any of the associated data will result in the calculated SHA256 value within the encrypted key block no longer matching the value produced during the key check process. Alteration of the data within the encrypted block itself would result in failure of the AESKW mode to properly decrypt the data.
e) The key block format and version are defined in the associated data, which is a plaintext value provided as part of the key block.
f) The key block supports AES. Other algorithms are supported by other key block formats within IBM systems.
g) The key wrapping method allows for the definition of the algorithm for the wrapped key, as does TR-31, and specific values for compliance with the PCI definition of ‘strong cryptography’ are noted in this report.
h) The AESKW mode of operation is defined in NIST SP800-38F.

Therefore, the IBM VLS algorithm can be seen as entirely equivalent to the key wrapping processes defined in TR-31 and ISO20038, in both security and equivalency criteria outlined by the PCI SSC.

The differences between the IBM VLS and TR-31 / ISO20038 can be best summarized as:

i. A different format is used for the metadata and key storage in the block.
ii. The VLS process binds the key parts together with SHA256 prior to encryption (which the TR-31 / ISO20038 process does not).
iii. The VLS process uses the AESKW mode of operation, rather than separate CBC and CMAC methods used for encryption and authentication in TR-31 / ISO20038.

Appendix I – IBM Control Vector Values

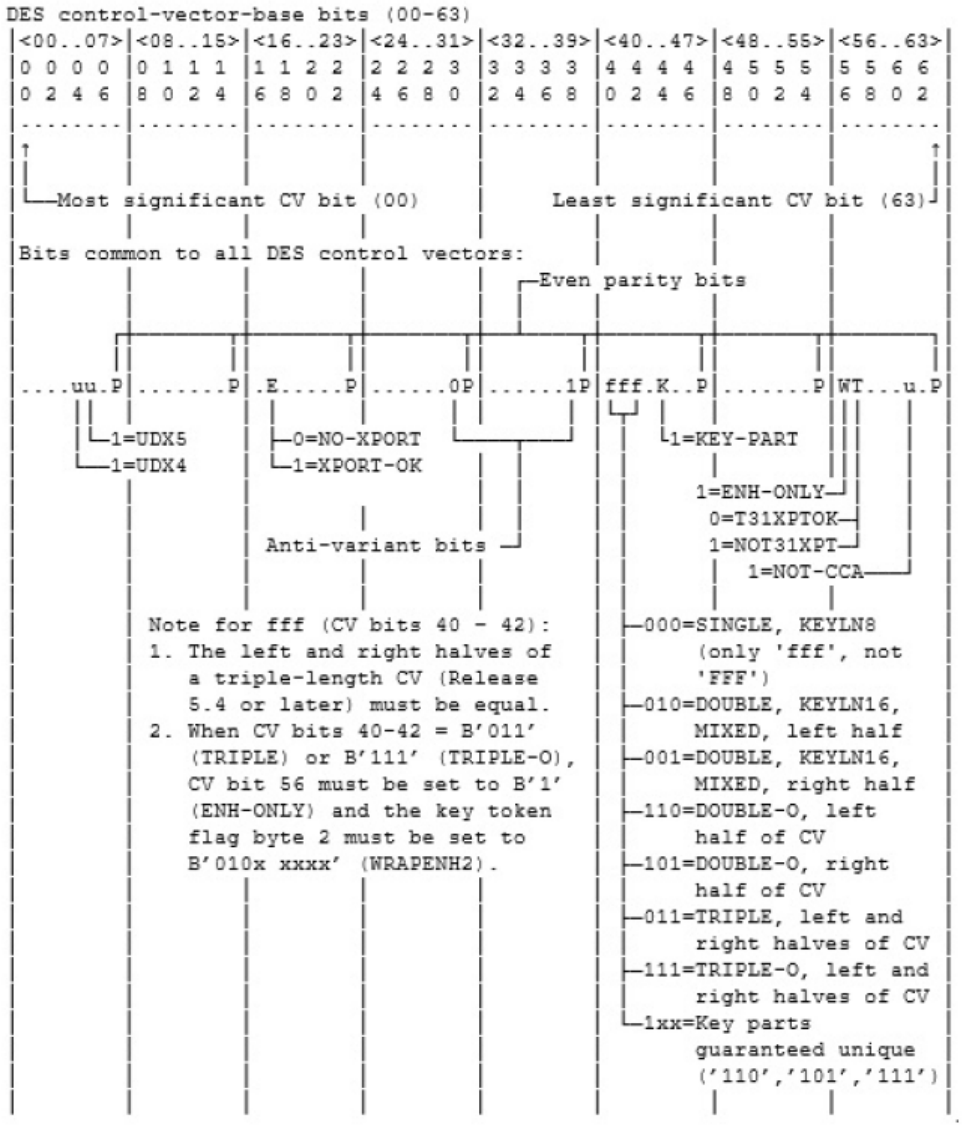

Figure 6 – IBM Control Vector 1 settings (1 of 4)

DES control-vector-base bits (00-63)

<00..07>	<08..15>	<16..23>	<24..31>	<32..39>	<40..47>	<48..55>	<56..63>
0 0 0 0	0 1 1 1	1 1 2 2	2 2 2 3	3 3 3 3	4 4 4 4	4 5 5 5	5 5 6 6
0 2 4 6	8 0 2 4	6 8 0 2	4 6 8 0	2 4 6 8	0 2 4 6	8 0 2 4	6 8 0 2
.....	.....	.....	.....	.....	.....	.....	.....
Data operation keys (bits 08-11 = B'0000'):							
		1=Encipher					
		1=Decipher					
		1=MAC generate					
		1=MAC verify					
DATA							
0000uu0P	00000000	0Eedmv0P	00000000	00000011	fff0K00P	00000000	WT000u0P
DATAC							
0000uu0P	00000000	0E11000P	00000000	00000011	FFF0K00P	00000000	WT000u0P
DATAM							
0000uu0P	00000000	0E00110P	00000000	00000011	FFF0K00P	00000000	WT000u0P
DATAMV							
0000uu0P	00000000	0E00010P	00000000	00000011	FFF0K00P	00000000	WT000u0P
CIPHER							
0000uu0P	00000011	0E11000P	00000000	00000011	fff0K00P	00000000	WT000u0P
DECIPHER							
0000uu0P	00000011	0E01000P	00000000	00000011	fff0K00P	00000000	WT000u0P
ENCIPHER							
0000uu0P	00000011	0E10000P	00000000	00000011	fff0K00P	00000000	WT000u0P
CIPHERXI							
0000uu0P	00001100	0E01000P	00000000	00000011	1FF0K00P	00000000	WT000u0P
CIPHERXO							
0000uu0P	00001100	0E10000P	00000000	00000011	1FF0K00P	00000000	WT000u0P
CIPHERXL							
0000uu0P	00001100	0E11000P	00000000	00000011	1FF0K00P	00000000	WT000u0P

Figure 7 – IBM Control Vector 1 settings (2 of 4)

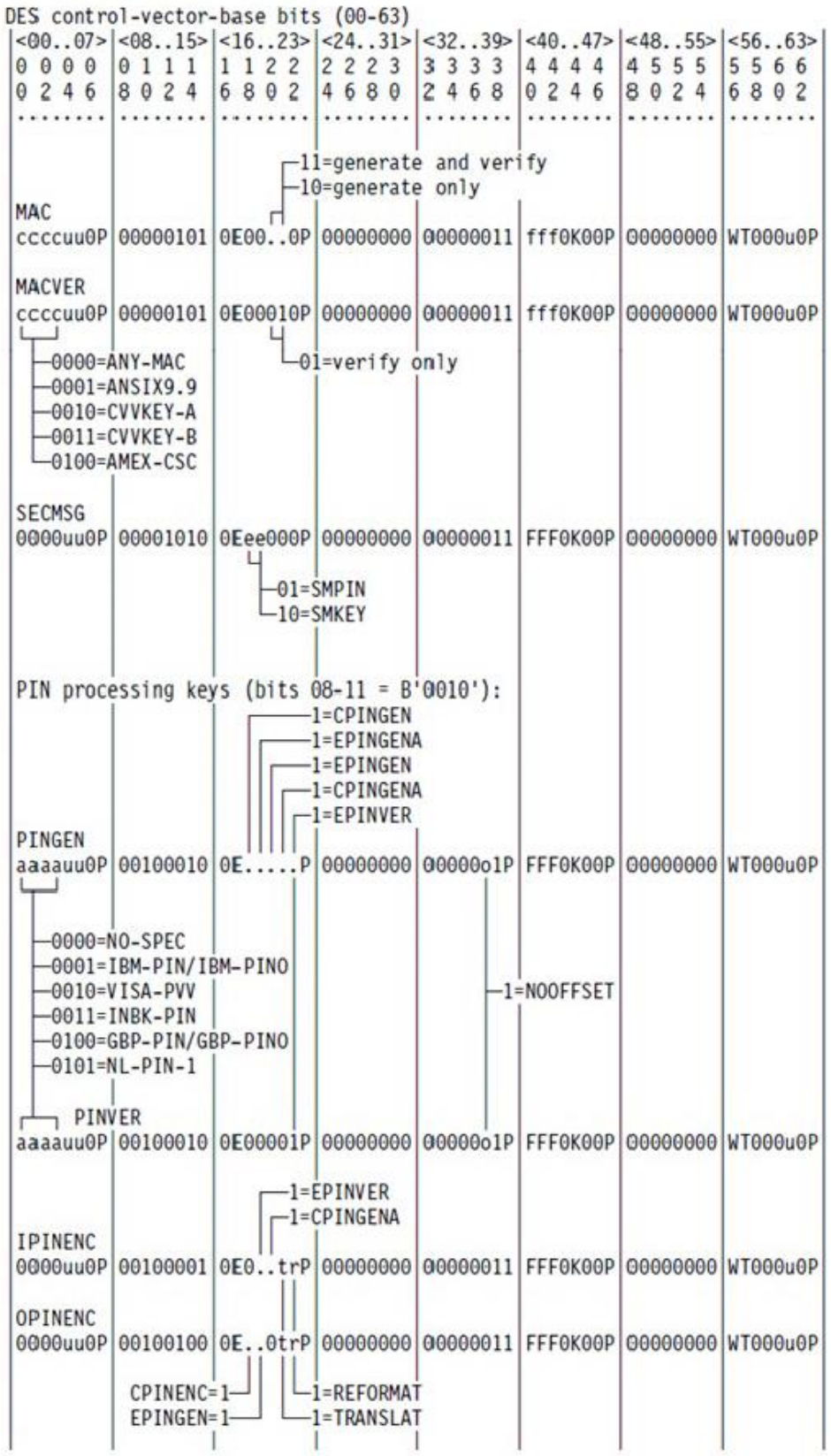

Figure 8 – IBM Control Vector 1 settings (3 of 4)

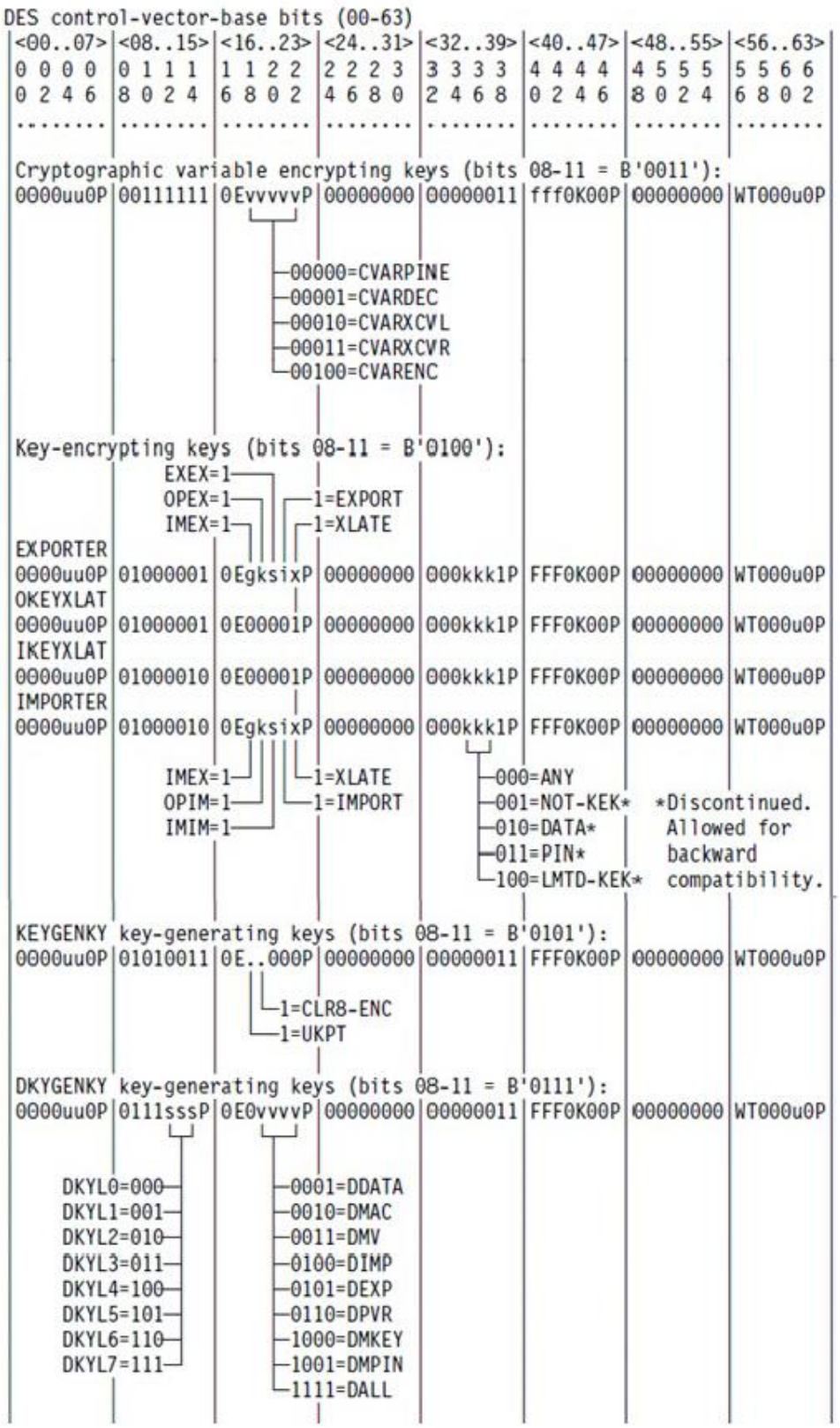

Figure 9 – IBM Control Vector 1 settings (4 of 4)

Appendix II – Authors

Andrew Jamieson

Education and Awards:

Bachelor of Engineering (Electrical Engineering, Honours)
Master of Applied Science (Information Security)
Distinguished Member of Technical Staff, UL
Adjunct Professor, New York University

Peer-reviewed Talks and Publications:

"Preventing cryptocurrency theft: Learning from the past to secure the future", Cyber Security; A Peer Reviewed Journal, 2019
"IoT Security, Building a Secure Future", Society of Instrumentation and Control Engineers (SICE) 2019 conference
"Biohazard! – Biometric Security", Auscert 2017
"IoT Security – It's in the Stars!", Auscert 2016
"Gone in a Flash!" (Flash memory security and data remnance), AusCERT 2012
"Securing Embedded Systems", AusCERT 2011
"Encryption vs Tokenisation", BSides Australia 2011

Patents :

"Keypad", International Patent WO0177801
"Circuit Board Arrangement and Funds Transaction Device", International Patent WO0189278
"Funds Transaction Terminal", International Patent WO0188862
"Display Device and Funds Transaction Devices Including the Display Device", International Patent WO03102785
"Secure Payment System", International Patent WO2005052801A1

Joehannes Bauer

Education and Awards:

Master of Computer Science (Honors)
Ph.D. of Computer Science (magna cum laude)

Peer-reviewed Talks and Publications:

Johannes Bauer: On Inexpensive Methods for Improving Security of Embedded Systems — Kostengünstige Maßnahmen zur Erhöhung der Sicherheit eingebetteter Systeme, November 2016. Doctoral Thesis. (URN urn:nbn:de:bvb:29-opus4-81273, 171 pages).
Johannes Bauer and Felix C. Freiling: Towards Cycle-Accurate Emulation of Cortex-M Code to Detect Timing Side Channels, August 2016. 11th IEEE International Conference on Availability, Reliability and Security—ARES 2016 (DOI 10.1109/ARES.2016.94, pages 49–58).
Johannes Bauer, Sebastian Schinzel, Felix C. Freiling and Andreas Dewald: Information Leakage behind the Curtain: Abusing Anti-EMI Features for Covert Communication, May 2016. IEEE International Symposium on Hardware Oriented Security and Trust—HOST 2016 (DOI 10.1109/HST.2016.7495570, pages 130–134).
Johannes Bauer, Michael Gruhn and Felix C. Freiling: Lest we forget: Cold-boot attacks on scrambled DDR3 memory, March 2016. Digital Forensics Research Workshop Europe 2016 (Elsevier Digital Investigation, Volume 16, Supplement, pages S65–S74). Open Access: http://www.sciencedirect.com/science/article/pii/S1742287616300032.
Johannes Bauer, Sebastian Schinzel, Felix C. Freiling and Andreas Dewald: Information Leakage behind the Curtain: Abusing Anti-EMI Features for Covert Communication, March 2016. Technical Report CS-2016-03 of the University Erlangen-Nürnberg (URN urn:nbn:de:bvb:29-opus4-71576).
Johannes Bauer and Felix C. Freiling: Schutz eingebetteter Systeme gegen physische Angriffe, September 2015. DACH Security 2015—Bestandsaufnahme – Konzepte – Anwendungen – Perspektiven (syssec-Verlag, Bonn, Germany, ISBN 978-3-000-49965-4, pages 387–396).
Johannes Bauer: Entwicklung einer OSEK/VDX-kompatiblen Systemschnittstelle für Linux, January 2008. Informatiktage 2008 (Lecture Notes in Informatics, ISBN 978-3-88579-440-0, pages 65–68).

Sajal Islam

Education and Awards:

Ph.D. of Computer Science, 2006, Monash University, Australia
B.Sc. Engineering - Computer Science & Engineering, 2002, Bangladesh University of Engineering & Technology, Bangladesh
CISSP, CISA, QSA, PA-QSA, P2PE QSA, P2PE PA-QSA, PCI 3DS Assessor, PCI PIN QPA

Peer-reviewed Talks and Publications:

Routing protocols for ad-hoc networks. Islam, M. M., Pose, R. & Kopp, C., 1 Dec 2007, Mobile Multimedia Communications: Concepts, Applications, and Challenges. IGI Global, p. 178-221 44 p.
Security in ad-hoc networks. Islam, M. M., Pose, R. & Kopp, C., 1 Dec 2007, Mobile Multimedia Communications: Concepts, Applications, and Challenges. IGI Global, p. 297-326 30 p.
Engineering a Suburban Ad-Hoc Network. Tyson, M. R., Pose, R. D., Kopp, C., Rokonuzzaman, SK. M. & Islam, M. M., 2006, Proceedings of the 7th Australian Information Warfare and Security Conference. Valli, C. & Woodward, A. (eds.). Perth WA Australia: School of Computer and Information Science, Edith Cowan University, p. 120 - 130 11 p.
An intrusion detection system for Suburban Ad-hoc Networks. Islam, M. M., Pose, R. D. & Kopp, C., 2005, Proceedings of the 2005 IEEE Region 10 Conference. Bradlow, H. (ed.). Melbourne Vic Australia: Swinburne Press, p. 1 - 6 6 p.
Challenges and a solution to support qos for real-time traffic in multi-hop ad-hoc networks. Islam, M. M., Pose, R. D. & Kopp, C., 2005, Proceedings of the Second IFIP International Conference on Wireless and Optical Communications Networks. Yeo, B. S., Ganaire, M. & Omidyar, C. G. (eds.). Dubai UAE: IEEE, Institute of Electrical and Electronics Engineers, p. 394 - 399 6 p.
Effects on directional antennas on 802.11e. Islam, M. M., Pose, R. D. & Kopp, C., 2005, Proceedings of the Second IFIP International Conference on Wireless and Optical Communications Networks. Yeo, B. S., Ganaire, M. & Omidyar, C. G. (eds.). Dubai UAE: IEEE, Institute of Electrical and Electronics Engineers, p. 1 - 6 6 p.
Link layer security for SAHN protocols. Islam, M. M., Pose, R. D. & Kopp, C., 2005, Proceedings of the Third IEEE Conference on Pervasive Computing and Communications Workshops on Pervasive Wireless Networking. Lee, B., Yu, C. & Mohapatra, P. (eds.). Los Alamitos USA: IEEE Computer Society, p. 279 - 283 5 p.
MAC layer support for real-time traffic in a SAHN. Islam, M. M., Pose, R. D. & Kopp, C., 2005, Proceedings of the International Conference on Information Technology: Coding and Computing. Selvaraj, H. & Srimani, P. K. (eds.). Los Alamitos USA: IEEE Computer Society, Vol. II. p. 639 - 645 7 p.
Making SAHN-MAC independent of single frequency channel and omnidirectional antennas. Islam, M. M., Pose, R. D. & Kopp, C., 2005, Proceedings of the IASTED International Conference on Networks and Communication Systems. Hamza, M. H., Prapinmonkolkarn, P. & Angakew, T. (eds.). Anaheim USA: ACTA Press, p. 220 - 225 6 p.
Suburban ad-hoc Networks in information warfare. Islam, M. M., Pose, R. D. & Kopp, C., 2005, Conference Proceedings of the 6th Australian Information Warfare and Security Conference. Pye, G. & Warren, M. (eds.). Geelong Vic Australia: School of Information Systems, Deakin University, p. 71 - 80 10 p.

A link layer security protocol for Suburban Ad-Hoc Networks. Islam, M. M., Pose, R. D. & Kopp, C., 2004, Proceedings of the Australian Telecommunication Networks and Applications Conference. Safaei, F. (ed.). NSW Australia: ATNAC, p. 174 - 177 4 p.
Multiple directional antennas in Suburban Ad-Hoc Networks. Islam, M. M., Pose, R. D. & Kopp, C., 2004, Proceedings of the International Conference on Information Technology: Coding and Computng. Srimani, P. K. (ed.). Los Alamitos USA: IEEE Computer Society, Vol. 2. p. 385 - 389 5 p.
A hybrid QoS routing strategy for Suburban Ad-Hoc Networks. Islam, M. M., Pose, R. D. & Kopp, C., 2003, Proceedings of the 11th IEEE International Conference on Networks. Moreton, N. (ed.). Piscataway NJ USA: IEEE, Institute of Electrical and Electronics Engineers, p. 225 - 230 6 p.
A router architecture to achieve link rate throughput in suburban ad-hoc networks. Islam, M. M., Pose, R. D. & Kopp, C., 2003, Proceedings of the 8th Asia-Pacific Conference in Advances in Computer Systems Architecture (ACSAC 2003). Omondi, A. & Sedukhin, S. G. (eds.). Berlin Germany: Springer-Verlag London Ltd., Vol. 2823. p. 395 - 407 13 p.
Efficient Routing in Suburban Ad-Hoc Networks (SAHN). Islam, M. M., Pose, R. D. & Kopp, C., 2003, Proceedings of the International Conference on Communications in Computing. Jd'Auriol, B. (ed.). USA: CSREA Press, p. 188 - 194 7 p.
Routing in Suburban Ad-Hoc Networks. Islam, M. M., Pose, R. D. & Kopp, C., 2003, Computer Science and its Applications: Proceedings of the International Conference on Computer Science and its Applications. Dey, P. P., Amin, M. N. & Gatton, T. M. (eds.). San Diego CA USA: US Educational Service LLC, p. 72 - 76 5 p.

Benoit Feix

Education and Awards:

Ph.D. Embedded Cryptography, Limoges University

Peer-reviewed Talks and Publications:

Benoit Feix, Andjy Ricart, Benjamin Timon, Lucille Tordella:
Defeating Embedded Cryptographic Protocols by Combining Second-Order with Brute Force. CARDIS 2016: 23-38
Julien Allibert, Benoit Feix, Georges Gagnerot, Ismael Kane, Hugues Thiebeauld, Tiana Razafindralambo:
Chicken or the Egg - Computational Data Attacks or Physical Attacks. IACR Cryptol. ePrint Arch. 2015: 1086 (2015)
Benoit Feix, Mylène Roussellet, Alexandre Venelli:
Side-Channel Analysis on Blinded Regular Scalar Multiplications. INDOCRYPT 2014: 3-20
Benoit Feix, Hugues Thiebeauld, Lucille Tordella:
Recovering CRT-RSA Secret Keys from Message Reduced Values with Side-Channel Analysis. INDOCRYPT 2014: 53-67
Benoit Feix, Mylène Roussellet, Alexandre Venelli:
Side-Channel Analysis on Blinded Regular Scalar Multiplications. IACR Cryptol. ePrint Arch. 2014: 191 (2014)
Benoit Feix, Hugues Thiebeauld:
Defeating ISO9797-1 MAC Algo 3 by Combining Side-Channel and Brute Force Techniques. IACR Cryptol. ePrint Arch. 2014: 702 (2014)
Benoit Feix, Alexandre Venelli:
Defeating with Fault Injection a Combined Attack Resistant Exponentiation. COSADE 2013: 32-45
Christophe Clavier, Benoit Feix:
Updated Recommendations for Blinded Exponentiation vs. Single Trace Analysis. COSADE 2013: 80-98
Benoit Feix, Vincent Verneuil:
There's Something about m-ary - Fixed-Point Scalar Multiplication Protected against Physical Attacks. INDOCRYPT 2013: 197-214
Christophe Clavier, Benoit Feix, Georges Gagnerot, Christophe Giraud, Mylène Roussellet, Vincent Verneuil:
ROSETTA for Single Trace Analysis. INDOCRYPT 2012: 140-155
Christophe Clavier, Benoit Feix, Loïc Thierry, Pascal Paillier:
Generating Provable Primes Efficiently on Embedded Devices. Public Key Cryptography 2012: 372-389
Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, Vincent Verneuil:
Improved Collision-Correlation Power Analysis on First Order Protected AES. CHES 2011: 49-62
Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, Vincent Verneuil:
Square Always Exponentiation. INDOCRYPT 2011: 40-57
Jean-Christophe Courrège, Benoit Feix, Mylène Roussellet:
Simple Power Analysis on Exponentiation Revisited. CARDIS 2010: 65-79

Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet:
Passive and Active Combined Attacks on AES???Combining Fault Attacks and Side Channel Analysis. FDTC 2010: 10-19
Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, Vincent Verneuil:
Horizontal Correlation Analysis on Exponentiation. ICICS 2010: 46-61
Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, Vincent Verneuil:
Horizontal Correlation Analysis on Exponentiation. IACR Cryptol. ePrint Arch. 2010: 394 (2010)
Frédéric Amiel, Benoit Feix, Michael Tunstall, Claire Whelan, William P. Marnane:
Distinguishing Multiplications from Squaring Operations. Selected Areas in Cryptography 2008: 346-360
Frédéric Amiel, Benoit Feix:
On the BRIP Algorithms Security for RSA. WISTP 2008: 136-149
Frédéric Amiel, Karine Villegas, Benoit Feix, Louis Marcel:
Passive and Active Combined Attacks: Combining Fault Attacks and Side Channel Analysis. FDTC 2007: 92-102
Frédéric Amiel, Benoit Feix, Karine Villegas:
Power Analysis for Secret Recovering and Reverse Engineering of Public Key Algorithms. Selected Areas in Cryptography 2007: 110-125
Bertrand Byramjee, Jean-Christophe Courrège, Benoit Feix:
Practical Attacks on Smart Cards. Handbook of Elliptic and Hyperelliptic Curve Cryptography 2005: 669-686

[i] ISO9564-1:1991, Banking – Personal Identification Number management and security – Part 1: PIN protection principles and techniques, https://www.iso.org/standard/17309.html
[ii] ISO11568-1:1995, Banking – Key management (retail) – Part 1: Introduction to key management
[iii] FIPS140-1, Security Requirements for Cryptographic Modules, https://csrc.nist.gov/csrc/media/publications/fips/140/1/archive/1994-01-11/documents/fips1401.pdf
[iv] ISO13491-1:1998, Banking – Secure Cryptographic devices (retail) – Part 1: Concepts, requirements and evaluation methods, https://www.iso.org/standard/19521.html
[v] [v] Deterministic Authenticated-Encryption – A Provable-Security Treatment of the Key-Wrap Problem, P Rogerway et al, https://www.iacr.org/archive/eurocrypt2006/40040377/40040377.pdf
[vi] IBM HSM Key Blocks, version 3.3.5.2
[vii] CCA Basic Services Reference and Guide for the IBM 4767 and IBM 4765 PCIe Cryptographic Coprocessors Releases 5.5, 5.4, 5.3, 4.4, and 4.2, Thirty Six Edition
[viii] NIST SP800-108, Recommendation for Key Derivation Using Pseudorandom Functions, https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-108.pdf
[ix] Symmetric Key Cryptography for The Financial Services Industry – Wrapping of Keys and Associated Data, ANSI X9.102 – 2008 (R2017), https://webstore.ansi.org/standards/ascx9/ansix91022008r2017
[x] Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping, SP800-38F, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf

[xii] [xii] PCI PTS PIN Information Supplement : Cryptographic Key Blocks, June 2017, PCI SSC, https://www.pcisecuritystandards.org/documents/Cryptographic_Key_Blocks_Information_Supplement_June_2017.pdf
[xiii] PCI PIN Requirements and Testing Procedures, Version 3.0, August 2018, PCI SSC, https://www.pcisecuritystandards.org/documents/PCI_PIN_Security_Requirements_Testing_v3_Aug2018.pdf

https://www.pcisecuritystandards.org/documents/Cryptographic_Key_Blocks_Information_Supplement_June_2017.pdf

[xv] PCI PIN Security Requirements Information Supplement: PIN Security Requirement 18-3 – Key Blocks, June 2019, PCI SSC, https://www.pcisecuritystandards.org/documents/PIN_Security_Rqmt_18-3_Key_Blocks_2019.pdf

[xvi] [xvi] See Q18 in PCI SSC PTS HSM Technical FAQs, PCI SSC, https://www.pcisecuritystandards.org/documents/PTS_HSM_Technical_FAQs_v3_September_2020.pdf