Technical note

Complementarity attacks and control vectors

by D. Longley
S. M. Matyas

A control vector is a data structure that specifies the nature and role of an associated cryptographic key. The control vector is checked by software and cryptographic hardware in order to limit the range of permissible operations to be undertaken with ciphertext produced with the key. The linking of the control vector and cryptographic key is such that attempts to modify, or substitute, control vectors will cause the subsequent processing to operate with a corrupted key, and hence ensure protection of data encrypted with the genuine key. A potential attack on the control vector approach is described in which the complement of the control vector is substituted. The manner in which such attacks are thwarted by the IBM implementation of control vectors is also described.

The paper "Key Handling with Control Vectors"[1] describes the use of control vectors associated with cryptographic keys and variables. The role of such control vectors is to restrict the cryptographic processing that may be undertaken with the cryptographic variables associated with the keys. It can be shown, however, that there exists a potential security flaw, in the control vector concept, that exploits the complementary property of the Data Encryption Algorithm (DEA).

In the first part of this note, D. Longley describes such an attack on a control vector system. In the second part, S. M. Matyas provides a detailed description of the IBM implementation of control vectors and indicates how the described potential attack is countered.

Attacks on key management schemes are often based upon misuse of key-encrypting keys. Typically ciphertext comprising a data key, encrypted by a key-encrypting key, is submitted to a cryptographic module for a data decryption operation in order to reveal the data key. A control vector may be associated with a key used to protect the confidentiality of a cryptographic key or data. This control vector provides information to system software and cryptographic hardware on the range of permissible operations for the associated cryptographic variables. Coupling of a control vector and the associated cryptographic key is designed to prevent a modified control vector from masquerading as the original when it is submitted for checking. If such a modified vector were submitted, the coupling between control vector and cryptographic key would ensure that corrupted keys are developed in the cryptographic hardware, and the subsequent output would be meaningless.

There is considerable flexibility in the definition of control vectors; the fields of the control vector are interpreted by software and cryptographic hardware, and system rules relate these fields to permitted processing by such software and hardware. The control vectors are coupled to the

cryptographic keys by a two-stage process. First, the variable length control vector (C) is subjected to a hashing function (h); the fixed-length hashed output ($h(C)$) is then added modulo two to the cryptographic key, within the cryptographic hardware.

This coupling of the control vector and key is designed to prevent substitution of a fake control vector. However, it is suggested that such a substitution of control vectors may reveal useful information to the attacker if the masquerading control vector is the complement of the original.

Complementarity attacks on the DEA

Complementarity attacks on the DEA are based upon the two identities.

For exclusive-OR operations:

$$A \oplus \bar{B} = \overline{(A \oplus B)}$$

and for DEA encryption:

$$e\bar{K}(\bar{P}) = \overline{(eK(P))}$$

In the case of a double-length key ($K1$, $K2$) and e-d-e encryption:

$$e\overline{K1}(\bar{P}) = \overline{(eK1(P))}$$

$$d\overline{K2}(\overline{eK1(P)}) = \overline{dK2(eK1(P))}$$

$$e\overline{K1}(\overline{dK2(eK1(P))}) = \overline{eK1(dK2(eK1(P)))}$$

The proposed attack scenario is based upon the postulated existence of two control vectors $C1$ and $C2$ such that

$$h(C1) = \overline{h(C2)}$$

The assumption that both $C1$ and $C2$ may be valid control vectors is discussed in the next section.

Assume that $h(C1)$ corresponds to the hashed control vector, where $C1$ is the control vector for a high-level key-encrypting key. $h(C2)$, which is the complement of $h(C1)$, is similarly postulated to be a hashed control vector, where $C2$ is the control vector for a data decipherment key.

Ciphertext concealing a key-encrypting key, KEK, is available in the form $eKC(KEK)$ where $KC = KK \oplus h(C1)$, and KK is the key-encrypting key used in association with the control vector. The key KEK in turn is employed to protect a data-encrypting key KD, so that $eKEK(KD)$ is also available. The object of the attack is to produce the plaintext key KD outside a cryptographic module.

The attacker first inverts $eKC(KEK)$ and $eKEK(KD)$ and submits $\overline{eKC(KEK)}$ with associated control vector $C2$, together with $\overline{eKEK(KD)}$ for a data decipherment operation. This procedure is valid since $C2$ is associated with a data decipherment key.

The key-encrypting key $KK \oplus h(C2)$ is formed and $\overline{eKC(KEK)}$ is first decrypted with $\overline{KC} = \overline{(KK \oplus h(C2))}$ to give:

$$d\overline{KC}(\overline{eKC(KEK)}) = d\overline{KC}(e\overline{KC}(\overline{KEK})) = \overline{KEK}$$

Next $\overline{KEK}$ is used to decrypt $\overline{eKEK(KD)}$ to give:

$$d\overline{KEK}(\overline{eKEK(KD)}) = d\overline{KEK}(e\overline{KEK}(\overline{KD})) = \overline{KD}$$

The attacker then inverts $\overline{KD}$ to reveal KD.

Key variants are always selected so that no one key variant is ever a complement to another, presumably to obviate the attack described above.

Control vectors specifications. The feasibility of the attack described above depends upon the fact that checking by system software and cryptographic hardware is performed using the control vectors themselves. However, the cryptographic coupling is based upon the hashed version for the control vectors.

As described in the paper on control vectors,[1] the hashing operations for 64-bit and 128-bit data structures are concatenation and identity functions, respectively, followed by the setting of the parity bits and the extension field. Thus there is an almost one-to-one relationship between the bits of the hashed control vector and the original data structure.

In this case the semantics of the control vector may be such that it is virtually impossible for an inverted hashed control vector to be of value to an

attacker. For example, inverting the antivariant field is likely to produce a requirement for the inverted hashed control vector to represent a key variant, which in turn places severe restrictions upon the permitted structure of the control vector.

However, in the case of data structures with lengths in excess of 128 bits, the hashing involves cryptographic one-way functions, and the hashed control vector has apparently no predefined structure, except for the parity bits and extension field. Thus there is very little relationship between the control vector used for checking authorizations, and the hashed control vector employed for cryptographic coupling.

It is therefore not inconceivable that the complement of the hashed control vector will, by chance, be accepted as a valid control vector by the cryptographic hardware, which in any case checks only a minority of the total set of bits comprising the control vector. The output of the hashing function is modified to include parity bits and a two-bit extension field indicating the length of the original data structure. Inversion of the seven data bits, and corresponding parity bit, will always produce correct parity. The extension field has two bits and three code values, hence inversion of two extension fields will produce a valid extension field. Interestingly enough, inversion of the extension field for data structures greater than 128 bits (B'10') produces a valid extension field for a 128-bit data structure (B'01'). This is convenient to the attacker, because the hashed version of the 64-bit control vector has a more defined structure, since it is formed by concatenation of the two 64-bit blocks.

In these circumstances there exists the possibility of the existence of two valid control vectors $C1$ and $C2$, with length of $C1 > 128$ bits and length of $C2 = 128$ bits such that $h(C2)$ is the complement of $h(C1)$. If such a pair is found, it will be possible to effectively operate upon the key associated with $C1$ using the control vector authorizations of $C2$ or vice versa, as described above.

It is, of course, assumed that both 128-bit data structures and those greater than 128 bits will be employed in the one system. However, the paper by Matyas[1] specifically mentions the use of the extension field to prevent a control vector derived from a data structure greater than 128 bits from being employed to masquerade as a control vector derived from a 128-bit data structure.

The attacker would first compute hashed functions for stored control vectors greater than 128 bits, compute their complements, and check if any corresponded to valid, and from the attacker viewpoint, useful control vectors for 128-bit data structures. It is shown below that, from a cryptographic control viewpoint, only a subset of control vector bits is used by the cryptographic hardware, and thus it is not necessary for the complement to involve all 128 bits. The reverse form of the attack, inverting hashed control vectors for 128-bit functions, is less likely to succeed, since the result would have to be matched against the hashed control vectors for existing control vectors greater than 128 bits, and there would have to be a one-to-one relationship between each of the 128 bits in the match. The one-way function used in the hashing operation prevents the attacker from generating the hashed control vector greater than 128 bits, thus restricting the search to stored control vectors greater than 128 bits.

A hashed 128-bit control vector has 18 fixed bits (16 parity plus two extension field bits), and hence, 110 bits are available for control checking. However, only a small proportion are used for checking by the cryptographic hardware; in the example given in the paper, only 17 bits are checked by the hardware (two of which will be valid extension field bits). Hence, the probability that an inverted hashed control value will be useful to an attacker is 1 in 32 768. This figure will be reduced by a factor according to the number of hashed control vectors greater than 128 bits available, and the number of variants on the resulting control vector that would be of value to the attacker. The computational effort is comparatively low; it involves the hashing function for all control vectors greater than 128 bits, inversion, and subsequent checking of the result against "useful" 128-bit control vectors.

In the above discussion it is assumed that there is no checking on the remaining 95 bits ($= 110 - 15$) of the control vector. In fact the Cryptographic Facility Access Program (CFAP) control subvector has been ignored, although it is employed as a check on the cryptographic operations to be undertaken when the control vector is submitted via the application programming interface (API). From the standpoint of the security of control

vectors, it is reasonable to argue that the security should be independent of implementation details outside the cryptographic hardware. The CFAP control subvector will be effective for high-level calls to the API, but an attacker may well be able to operate at a lower level, or at least arrange for the control vector, submitted to the cryptographic hardware, to differ from that submitted for CFAP control subvector checking. Indeed one of the advantages of the control vector is that much of the checking may be conducted independently of the various subvectors.

The defense against the attack is fortunately very straightforward: simply use three bits for the extension field and, of the eight possible sets of values, set three that do not correspond to inversions of a valid field, e.g., 000, 010, and 001. The first bit of the extension field provides a guarantee that inversion will produce an invalid extension field. The extension field provides the only independent bits that are set after the hashing function and is therefore the only one that may be used for this purpose.

IBM implementation of the control vector

Complementarity attacks were indeed anticipated and blocked in the IBM implementation of the control vector. This aspect of the control vector design was not discussed in the paper by Matyas.[1]

In an early design of the control vector,[2] complementarity attacks were thwarted by using a four-bit extension field, along the lines suggested by Longley. The three control vector (CV) forms were represented as B'0000' for 64-bit CV, B'0001' for 128-bit CV, and B'0010' for CV greater than 128 bits. However, a different approach was taken in the final design.

During the design phase, it was recognized that control-vector-based systems might need to coexist with variant-based systems, and therefore the control vector design should prevent keys encrypted with variants from being substituted for keys encrypted with control vectors (and vice versa). Thus, a general design was sought that would satisfy the following design objectives:

1. Separation of control vectors of one form (say, 64-bit) from control vectors of another form (say, 128-bit)
2. Separation of control vectors from IBM variants
3. Separation of control vectors from complements of control vectors

The three design objectives are achieved through an appropriate specification of hashing function h.

The hashing function described in Reference 1 was somewhat oversimplified. In the actual IBM implementation, the hashing function h also calls for the setting in $h(C)$ of several additional bits, including two antivariant bits. The antivariant property is achieved (1) by selecting two bit locations, i and j, in $h(C)$, such that their difference $(j - i)$ is a multiple of eight, and (2) by fixing the value of one bit to zero and the other to one. In the IBM implementation, the antivariant bits are specified at bit locations 30 and 38 and have fixed values B'0' and B'1', respectively. Since a 64-bit IBM variant mask value is formed by replicating a single eight-bit value eight times, this forces the bits in locations 30 and 38 of the variant mask value to be equal. Thus, the desired separation between control vectors and variants is achieved. Moreover, since the antivariant bits are a constant value (B'01'), an inversion of the field produces an invalid value (B'10'). Such an invalid value will be detected as part of the control vector checking process, thus preventing attacks based on complementarity.

The complete specification for setting of bits in the hash value $h(C)$ can be summarized as follows:

1. Set antivariant field (bits 30, 38) as follows:
   a. Set bit 30 equal to B'0'
   b. Set bit 38 equal to B'1'
2. Set extension field (bits 45, 46) as follows:
   a. B'00' if C is 64-bit
   b. B'01' if C is 128-bit
   c. B'10' if C is >128-bit
3. Set key/key-authenticator field (bit 62) as follows:
   a. B'0' if C is associated with a key value
   b. B'1' if C is associated with a key authenticator
4. Set parity bits 7, 15, . . . , 127 to even parity

The final design has the advantage of having complementarity attacks defended against without requiring the specification of a separate anticom-

plement bit, which, for practical purposes, is the net effect of using a three- or four-bit extension field instead of a two-bit extension field. Axiomatically, one does not want to waste control vector bits, especially for short (64-bit) control vectors.

The present specification of the antivariant field has the disadvantage of highlighting the antivariant property but not highlighting the anticomplement property. A more self-defining specification can be achieved by changing the field name, antivariant, to antivariant/anticomplement, or by redefining bit 30 as anticomplement zero (constant B'0') and bit 38 as antivariant one (constant B'1'), respectively. In the latter case, the anticomplement zero field ensures separation of control vectors from complement control vectors, and the antivariant one and anticomplement zero fields, taken together, ensure separation of control vectors from variants.

Summary

Control vectors are designed to limit the cryptographic processing that may be performed with ciphertext variables. The control vector is examined by system software and the cryptographic hardware to check authorization; the control vector is also coupled with cryptographic keys to prevent the substitution of a fake control vector. The form of coupling involves bitwise modulo two addition of the control vector and key. This technical note described a potential form of attack in which a complement of the control vector is exploited. It concludes with a detailed description of the IBM implementation of control vectors that was designed to obviate such attacks.

Acknowledgments

The subject of this note originated with Dennis Longley while he held a Visiting Fellowship in the Department of Computer Science, City Polytechnic of Hong Kong. He wishes to express his thanks to Dr. Balasubramanian and his colleagues for the facilities and help provided that enabled him to complete work on the note.

Cited references

1. S. M. Matyas, "Key Handling with Control Vectors," *IBM Systems Journal* **30**, No. 2, 151–174 (1991).
2. S. M. Matyas et al., *Secure Management of Keys Using Extended Control Vectors*, U.S. Patent No. 4,924,515, issued May 8, 1990.

Accepted for publication December 1, 1992.

Dennis Longley *Information Security Research Centre, Queensland University of Technology, GPO Box 2434, Brisbane, Queensland 4001, Australia.* Professor Longley is Dean of Faculty of Information Technology, Queensland University of Technology. Prior to his current post he was Head of the Department of Computing and Cybernetics, Brighton Polytechnic, England. In 1988, he founded the Information Security Research Centre, which is now a major research team in the Distributed Systems Technology Centre funded by the Commonwealth Government. He has published research papers in key management and, with coauthors, three books, *Dictionary of Data and Computer Security, Information Security for Managers*, and *Information Security Handbook*. He has acted as information security consultant to the Royal Hong Kong Jockey Club, a German Banking Organization, SKO, the Gold Casket Art Union, and Eracom Pty Ltd.

Stephen M. Matyas *IBM Federal Systems Company, 9500 Godwin Drive, Manassas, Virginia 22110.* Formerly a member of the Cryptography Competency Center at IBM's Kingston development laboratory, Dr. Matyas is currently manager of the Secure Products and Systems group at Manassas, Virginia. He participated in the design and development of all major IBM cryptographic products. He played a lead role in the design of IBM's Common Cryptographic Architecture and is the inventor of the control vector concept. Dr. Matyas holds 28 cryptographic patents and has published numerous technical articles covering many aspects of cryptographic system design. He is the coauthor of an award-winning book, *Cryptography—A New Dimension in Computer Data Security*, and is a contributing author to the *Encyclopedia of Science and Technology* and to *Telecommunications in the U.S.—Trends and Policies*. Dr. Matyas received a B.S. degree in mathematics from Western Michigan University and a Ph.D. degree in computer science from the University of Iowa. He is the recipient of an IBM Outstanding Innovation Award and an IBM FSC President's Patent Award, and is an 18th level inventor. Dr. Matyas is presently an IBM Senior Technical Staff Member.

Reprint Order No. G321-5516.