A Chosen Key Difference Attack on Control Vectors

M.K. Bond

1st November 2000

Abstract

An attack on the implementation of control vectors in the IBM Common Cryptographic Architecture is presented. The final key−part holder in a multiple part import introduces two *key−encrypting keys* (KEKs), one the intended key and one with a chosen difference from the former, by including this difference in his own key part. When this difference is set to the difference between two control vectors, keys originally encrypted with the former KEK can be cast to a new type by importing them under the latter KEK. Thus unauthorised type−casts can be made from an arbitrary source type to any destination type the attacker has permission to use.

1 Introduction

Cryptoprocessors store large quantities of key material securely by protecting a single secret − the *master key* − and encrypting other keys with this. These encrypted keys can then be stored in quantity on the hard drive of the host computer. When cryptoprocessors with different master keys need to communicate key information securely, the operators manually set up *key−encrypting−keys* (KEKs) to protect other keys during transport. Import and export transactions convert keys from encryption with a KEK to the local master key, and vice−versa.

The same encryption algorithm is used for many transaction types, so the key type must be verified as appropriate to the transaction. This is to ensure that a role−based access control system, with granularity only down to permit/deny per transaction, is sufficient to implement the security policy.

The example in this paper focuses on secure banking systems, using the CCA as implemented on the IBM 4758 cryptoprocessor. The data and PIN processing keys are processed with the same algorithm, normally triple−DES. *Data keys* are used for regular encryption of confidential bank traffic and are thus available to many roles. *PIN processing keys* permit calculation of pin numbers from publicly available account information, thus transactions using them should be restricted to authorised roles within the bank. If a PIN key could be re−typed as a data key, any role with access to data manipulation transactions could calculate the PIN numbers from accounts.

S. Matyas describes the implementation of the control vector system in [2], to allow advanced key typing within the CCA. Section 2 gives a brief summary of control vectors.

2 Control Vectors

Control vectors are the CCA name for key type information. The simplest method of binding the information is to append the type to the key before encryption under the master key Km − $E_{Km}(KEY , TYPE)$.

However, the CCA binds the information by XORing the control vector with the key used for encryption, and appending the control vector in the clear :

$$E_{Km \oplus TYPE}(KEY), TYPE$$

This has the advantage of keeping the length of the encrypted portion constant, yielding performance increase, and assisting backwards compatibility with earlier IBM CCA products. If a naive attacker changes the claimed type of the key (i.e. the clear copy of the control vector), when the key is used, the cryptoprocessor's decryption operation should simply produce garbage:

$$D_{Km \oplus TYPE\_MOD}(E_{Km \oplus TYPE}(KEY)) \neq KEY$$

3 The Attack

3.1 Obtaining keys with a chosen difference

KEKs are often transferred in multiple parts, which are combined using XOR to form the final key. If the key parts are given to separate staff, a dual control policy can be implemented, requiring collusion to reveal the value of the key. However, any of the key part holders could modify his part at will, and thus introduce a known difference between the intended key value, and the actual value loaded. For illustration, imagine the loading of a KEK denoted KORIG, from three parts KPA, KPB, KPC. The attack requires the final key part holder (KPC) to repeat the final step a second time, introducing a bogus KEK denoted KMOD, by entering his key part XORed with a modification pattern CVMOD.

$$\texttt{KORIG} = \texttt{KPA} \oplus \texttt{KPB} \oplus \texttt{KPC}$$
$$\texttt{KMOD} = (\texttt{KORIG} \oplus \texttt{CVMOD}) = (\texttt{KPA} \oplus \texttt{KPB} \oplus (\texttt{KPC} \oplus \texttt{CVMOD}))$$

Note that the purpose of introducing the true key is only to allow normal imports to occur, preventing the arousal of suspicion.

3.2 An unauthorised type cast

Now let another cryptoprocessor transmit a PIN processing key PKEY, protected using KORIG, the true KEK. The key type − *pinkey* − is bound in the usual way. The attacker now changes the claimed type to *datakey* and imports it under the bogus KMOD. The modification pattern CVMOD is set to $(\textit{pinkey} \oplus \textit{datakey})$.

$$E_{\texttt{KORIG}\oplus \textit{pinkey}}(\ \texttt{PKEY}\ ),\ \textit{pinkey} \quad \rightarrow \quad E_{\texttt{KORIG}\oplus \textit{pinkey}}(\ \texttt{PKEY}\ ),\ \textit{datakey}$$

The identity $\texttt{KORIG}\oplus \textit{pinkey} = (\texttt{KORIG}\oplus \textit{pinkey}\oplus \textit{datakey})\oplus \textit{datakey}$ causes the cyptoprocessor to retrieves PKEY correctly, which it then re−encrypts under the local master key, re−binding the false type *datakey*.

$$D_{\texttt{KMOD}\oplus \textit{datakey}}(\ E_{\texttt{KORIG}\oplus \textit{pinkey}}(\ \texttt{PKEY}\ )\ ) = \texttt{PKEY}$$
$$\rightarrow\ E_{\texttt{Km}\oplus \textit{datakey}}(\ \texttt{PKEY}\ )$$

4 Implementation Specific Issues

The CCA commands used by this attack are `Key_Part_Import` and `Key_Import`, as detailed in [1]. Various complications arise depending on whether the key parts are accumulated in an internal key register or in a public token, and dependent upon the physical transport method. Note that internal key part accumulation is not a problem if the attacker has the option to admit the bogus KEK on the first pass, claim a typing error occurred, and then admit the true KEK on the second attempt. Transferring key material using smartcards and 'trusted key entry' workstations can make this sort of tampering much more difficult.

The attack also relies on the optional nature of key integrity testing using `Key_Test`, which is designed to detect accidental errors in entry. However, if the requirement to pass this test is enforced in the cryptoprocessor firmware, then the bogus key cannot be admitted.

For simplicity, this paper has shown the control vectors being directly XORed with the encryption key, but the CCA implementation [2] XORs the key with the hash of the control vector, using a special hash function. However, as the possible control vectors values are publicly defined constants, this hash function should be considered reversible, and thus does not affect the attack.

5 Conclusions

The attack presented exploits the group nature of the XOR operator, the function at the heart of the control vector & key binding process. A possible solution is to hash the control vector together with the key, instead of using XOR, but this modification would damage backwards compatibility with IBM cryptoprocessors using the 'variants' scheme (the predecessor to control vectors).

A standard cryptoprocessor design rule is that *"No user should ever be able to obtain the value of a key within the system"*. If the binding method must continue to use XOR, it appears that in addition to this, an extra condition should be added : *"No user should ever be able to know the difference between keys within the system"*. It remains to be seen whether the CCA is capable of meeting the requirements of this second rule without seriously damaging its functionality.

The control vector method of controlling key usage differs significantly from other methods in that it cannot always halt the progress of a transaction if the type information is inconsistent, thus permitting transactions to produce "garbage" results. In [2], S. Matyas notes that *"a good key management design will ensure that such spurious keys are of no beneficial use to a would−be adversary."*, but the consequences of this allocation of responsibility is to intimately interlock the security of the transaction set with the details of its application. This interlock occurs at a detail level much finer than the application programmers are led to believe. The attack was discovered during ongoing research into formal verification methods for transaction sets, and this goal will likewise suffer great difficulty whenever this fine grain dependency is present.

6 Acknowledgements

The author wishes to thank (alphabetically) Ross Anderson, George Danezis & Larry Paulson for their assistance in verifying and understanding the consequences of this attack.

7 References

[1] IBM 4758 PCI Cryptographic Coprocessor, CCA Basic Services Reference And Guide, *Release 1.31 for the IBM 4758−001*

[2] S.M. Matyas, "Key Handling with Control Vectors", *IBM Systems Journal v. 30 n. 2, 1991, p. 151−174*

[3] S.M. Matyas, A.V. Le, D.G. Abraham, "A Key Management Scheme Based on Control Vectors", *IBM Systems Journal v. 30 n. 2, 1991, pp. 175−191*

[4] B. Schneier, "Applied Cryptography" *2nd Edition, John Wiley & Sons Inc, p. 561, 180*

[5] Transaction Security System, Basic CCA Cryptographic Services, *First Edition July 1997*

[6] D. Longley, S. M. Matyas, "Complementarity Attacks and Control Vectors", *IBM Systems Journal v. 32 n.2, 1993, p. 321−325*