IBM Comment on "A Chosen Key Difference Attack on Control Vectors"

Thank you for providing us with a copy of your paper "A Chosen Key Difference Attack on Control Vectors". This document contains IBM's comments on that paper, and implementation guidelines to help customers avoid the problem.

Introduction

The paper describes a method of changing the Control Vector (CV) of a CCA[1] key by modifying the value of the Key Encrypting Key (KEK) that is used to encipher or decipher that key. In this method, the KEK is altered by exclusive-ORing it with both the original CV and the desired new CV.

$$K_{NEW} = K_{ORIGINAL} \oplus CV_{ORIGINAL} \oplus CV_{DESIRED}$$

This method is well known in the CCA community, where it is called the "Pre-exclusive-OR Technique". It is described in Appendix C of the 4758 CCA Basic Services manual, under the heading *"Changing Control Vectors with the Pre-Exclusive-OR Technique"*[2]. If used as described in the Bond paper, it can be viewed as an attack. When used in another context, however, it is a valuable feature that facilitates key interchange between CCA and non-CCA cryptographic systems. Customer applications use this method to attach a Control Vector to a key received from a non-CCA system that does not employ Control Vectors. Appendix A on page 5 shows an example of this process.

The IBM designers recognized that the functions that import cleartext key parts are sensitive and subject to misuse by dishonest users. For this reason, the system was designed so that these functions can be restricted to authorized systems personnel, and not permitted for general users.

Two properties are desirable for the process of cleartext key import:

1. If there are *n* key parts, all *n* people must collude in order to disclose the value of the key. No useful information about the key's value can be obtained by fewer than *n* people working together.

2. No single person should be able to alter the final key value in a way that subverts security of the system.

As Bond demonstrates, the *Key_Part_Import* verb by itself does not accomplish the second of these goals. Additional measures are needed in order to ensure that this goal can be met.

Summary of the attack

In Bond's attack, two separate but related Key Encrypting Keys are installed in the system using the *Key_Part_Import* verb. The first is the correct, intended key, denoted KORIG. The second key is a modified version of the intended key, with the value KORIG⊕CVMOD, where CVMOD is the exclusive-OR of two control vectors, CV1 and CV2. This modified version of the KEK is called KMOD.

At a later time, the system receives a key K, enciphered under key KORIG. The key K has an associated key type, defined in its control vector CV1. If K is imported with the *Key_Import* verb using KORIG, it retains the control

[1] CCA is the IBM Common Cryptographic Architecture. The CCA implementation for the IBM 4758 PCI Cryptographic Coprocessor is the subject of Bond's paper.

[2] The Basic Services manual is available on the 4758 web site at http://www.ibm.com/security/cryptocards. Click on "Library", then click on the most recent version of the "CCA Basic Services" manual.

vector CV1. If it is imported using the modified KEK KMOD, however, the control vector for the key K is translated from CV1 to CV2.

CV1 defined the permitted usage of key K, and by altering the control vector to CV2, the attacker has changed the operations for which K can be used. In Bond's example, CV1 defines the key as a PIN key, which can only be used in special PIN processing functions. CV2, however, is for a DATA key, which allows the key to be used in general-purpose data encipher and decipher functions. This would allow the attacker to use K to decrypt PIN blocks that were encrypted under the key.

This attack is not restricted to the scenario in which a PIN key is converted to a DATA key. For example, an attacker could instead redefine a Key Encrypting Key as a double-length DATA key. The redefined key could be used to decrypt all received keys that were encrypted with that KEK.

Bond's attack requires the dishonest individual to be the person who enters the last part of a multi-part key. This is the only practical way for the attacker to produce two versions of the final key, KORIG and KMOD. Although it is more difficult, a similar attack is also possible by a person who enters a key part other than the last part. If this person exclusive-ORs his key part with CVMOD, the result after the last key part is entered will be a single version of the final key, but that version will be KMOD and not KORIG. This means that the attacker's key will be in the system, and it can be used for the attack. It also means, however, that the attack is likely to be detected, if the new control vector CV2 causes an error when the key is used by the real application program.

Recommendations

This section provides recommendations on CCA application design and administrative procedures, related to the Chosen Key Difference Attack.

Note that Bond's attack is a one-party attack. The recommendatoins below assume that this is the case, and that multiple attackers are not colluding to obtain the cryptographic keys.

Recommendation 1: Use public-key techniques instead of cleartext key part entry

The use of cleartext key parts is not a recommended approach in today's world, but it was the standard method for many years, and must be supported to meet the needs of those customers who still prefer it. With CCA, a preferable alternative is the use of the *Symmetric_Key_Import* verb, which receives a KEK encrypted with the receiver's RSA public key. With this approach, the KEK value never appears in the clear, and it is not subject to any attacks against its Control Vector. This is commonly used in one of two ways.

1. Encrypt the desired KEK under the recipient's RSA public key, using the *Symmetric_Key_Export* verb[3], and send the encrypted key to the recipient. That person decrypts the key using the *Symmetric_Key_Import* verb with their private key, and then uses the recovered KEK for the desired cryptographic operations.

2. Use the *PKA_Symmetric_Key_Generate* verb to create a randomly-generated KEK. The verb simultaneously produces two copies of the key, in different forms. One copy is encrypted under your local master key for use in your cryptographic environment. The other copy is encrypted under the recipient's RSA public key. Send this second version to the recipient, who decrypts the key using the *Symmetric_Key_Import* verb with their RSA private key. The two of you can then exchange other KEKs encrypted under this common, randomly-generated KEK, and you use those keys in the desired cryptographic operations such as PIN processing.

Recommendation 2: Use the access control system

The 4758 has an integrated role-based access control system that controls a user's authority to execute CCA verbs. In addition, the S/390 (zSeries) and AS/400 (iSeries) systems each have their own access control systems which are

[3] This could also be done using other functions, on products that do not use CCA.

able to control use of the verbs. Many security problems can be prevented by carefully dividing CCA verb execution rights among the users, and by prohibiting the use of selected verbs.

If the *Symmetric_Key_Import* verb will be used to receive Key Encrypting Keys, as suggested in Recommendation 1 above, the access control system can be used to block the chosen key difference attack described in the paper. If you configure the access control system to disable the *Key_Part_Import* functions, no user will be allowed to execute that verb. This prevents any loading of cleartext key parts, thus making the attack impossible.

The access control system can also be used to eliminate the chosen key difference attack, when importing cleartext key parts as described by Bond. The attack scenario requires the adversary to execute two separate CCA verbs.

- *Key_Part_Import* is used to create the modified Key Encrypting Key, designated KMOD in the paper.
- *Key_Import* is used with key KMOD to receive a PIN key and alter its Control Vector.

If no single person has the authority to execute both *Key_Part_Import* and *Key_Import*, then it will be impossible for any one person to mount the attack. It would require collusion of two people, who would each have authority to execute only one of these CCA verbs. Thus, the authority to execute these two verbs should be assigned to different people.[4] Appendix B on page 6 shows an example of this configuration and its use.

Recommendation 3: Use procedural and environmental controls

It is important to consider the environment in which keys are entered into a system. This should be done in a controlled-access environment, by trusted individuals. Within this environment, supervision and procedural controls can be used to reduce the likelihood of the attack.

Procedures can be implemented that require two people to be present whenever keys are imported. For maximum effectiveness, each of the two people should separately receive their own copy of the key part that is to be entered. One person is responsible for entering the key part, and the second person verifies that the entered data matches what is on his copy of the key part data. Audit logs can also be collected and reviewed. For example, each key entry operation can be monitored and logged, so that any unapproved actions can be traced to the responsible individual.

As the paper suggests, mandatory checking of the key verification pattern can be another solution. The person who generates the key can send the key verification pattern to someone other than the people who will enter the key parts. After the last key part is entered, this person uses the *Key_Test* verb to generate the verification pattern for the new key, and then compares the generated verification pattern to the one he received from the key originator. If the two do not match, the key would be immediately deleted. (Note that this approach is not possible if the key parts are generated separately, and never combined into the final key until installation into the cryptographic device. In this situation, there is no way to compute the verification pattern before the key is installed.) Appendix B on page 6 shows an example of this process in more detail.

The attacker must also have the authority and opportunity to intercept key and PIN data being used by the live application program, in order to make use of the modified key. Generally, the security officer responsible for key entry will not also be a system programmer, with access to the skills and tools required to tamper with the application program and its data. Procedures that control access to the machine and its files can also be used to ensure that one person cannot do all of these things that are required for the complete attack.

[4] This is actually a simplification. The *Key_Part_Import* verb has two separately authorized sub-functions: Entry of the first key part, and entry of the intermediate or last key parts. Authority to execute these sub-functions should also be assigned to separate people.

Conclusion

To summarize the points above:

- IBM agrees that the attack is viable under specific circumstances, but the method used is well-known and well-understood, and the required circumstances are unlikely in a carefully designed and deployed application.
- The method described in the paper has useful properties that allow import of keys from non-CCA systems. The CCA manual describes the method and its use for this purpose.
- We recommend using public-key techniques to interchange these keys, instead of using cleartext key parts.
- The access control system can be used to reduce or eliminate the risk.
- Key entry should be done by fully-trusted individuals.
- The environment makes the attack difficult in most cases, and procedural measures can reduce the risk.

We appreciate the time and effort you put into your analysis, and we look forward to any other observations you make as you continue this work. As a result of this paper, we intend to expand our documentation to describe this exposure, and to describe the various methods that should be used to ensure it does not result in a weakness in the customer's system.

Appendix A: Receiving a non-CCA key into a CCA system

This section shows an example of the technique used to import a DES key from a non-CCA system into CCA, adding a Control Vector in order to produce a valid CCA internal key token.

Assume the sender encrypts a single-length DES key K1 using Key Encrypting Key KEK1, with no CV. You receive the value:

$E_{KEK1}(K)$

In order to use key K in your CCA system, you need to associate the key with an appropriate CV, and then use the key and CV to produce a CCA internal key token in the form:

$\{ E_{KM \oplus MYCV}(K) , MYCV \}$

where MYCV is the Control Vector you want to associate with key K.

At an earlier time, assume you have received the cleartext key KEK1 from the sender, either as a single value, or in multiple parts. On your system, you produce a key KEK1MOD by the method described in the paper, as follows:

KEK1MOD = KEK1⊕MYCV
Import KEK1MOD using the *Key_Import* CCA verb

Now, when you receive the encrypted key $E_{KEK1}(K)$, you import it into your system using the modified key KEK1MOD and the Control Vector MYCV. CCA performs the following steps to recover the key.

$D_{KEK1MOD} (E_{KEK1} (K)), MYCV$
$= D_{KEK1 \oplus MYCV} (E_{KEK1} (K)), MYCV$
$= D_{KEK1 \oplus MYCV \oplus MYCV} (E_{KEK1} (K)), MYCV$ (CCA XORs in the key's CV as part of the key unwrapping)
$= D_{KEK1} (E_{KEK1} (K)), MYCV$
$= K, MYCV$

This is encrypted under the master key, to produce the desired key token, $\{ E_{KM \oplus MYCV} (K) , MYCV \}$.

The result is that you now have a CCA internal key token, which allows you to use key K in your CCA system under the restrictions defined in the Control Vector MYCV.

Note that this method is more complex if the key K is double-length. In this case, you must create two separate modified Key Encrypting Keys, rather than the one key KEK1MOD. Each modified KEK is used to import one of the halves of key K.

Appendix B: Sample process and configuration using cleartext KEKs

This section shows an example of how to improve security when using cleartext Key Encrypting Keys (KEKs). The example shows how to divide responsibility among several people, using the 4758 access control system and procedural measures.

This procedure involves four separate people. The table below describes their responsibilities, and the authorities they have under the 4758 access control system at the machine receiving the keys. The configuration at the systems that generate and export the keys is outside the scope of this paper.

Identifier	Responsibilities	Access control authorizations at target 4758
A	Generates and distributes cleartext KEK parts, and verification pattern (VP) for the KEK.	N/A
B	Enters first part of KEK into 4758.	Permitted to load first key part.
C	Enters second part of KEK into 4758, and verifies completed key is correct by checking verification pattern.	Permitted to load last key part, and to calculate the key verification pattern.
D	Distributes a PIN key, encrypted under the KEK.	N/A
E	Verifies KEK is correct, and imports PIN key that is encrypted under the KEK.	Permitted to calculate the KEK verification pattern, and to import the PIN key.

These people perform the following steps in order to securely import a two-part cleartext KEK, and then import a PIN key encrypted under this KEK.

1. Person *A* generates the two cleartext components of the KEK. He combines the parts to form the complete key, and calculates the verification pattern (VP) on the key.

   He sends the first key part, KP1, to person *B*.

   He sends the second key part, KP2, and the verification pattern VP to person *C*.

   He sends the verification pattern, VP, to person *E*.

   If person *E* is not in the same location as person *A*, the KEK must also be transmitted to person *E* for later use. This could be done using cleartext key parts, using an existing shared key encrypting key, or using an RSA key pair. In some environments, person *A* and person *E* will be the same person, and this step will not be necessary.

2. Person *B* enters key part KP1 into the 4758.

3. Person *C* combines key part KP2 into the value KP1 left by person *B* in step 2. This produces the final key, KEK.

   After the key is complete, person *C* verifies that the key has the correct value. He uses the CCA *Key_Test* verb to calculate the verification pattern on KEK, and compares this result with the verification pattern VP he received from person *A* in step 1. If the two do not match, an error has occurred, and he deletes the KEK from the system to prevent it from being used.

4. Person *D* exports the PIN key under the KEK, and sends the encrypted PIN key to person *E*.

5. Person *E* verifies that the KEK has the correct value by using *Key_Test* to calculate the verification pattern, and comparing it to the verification pattern VP received in step 1. If the two verification patterns do not match, the

KEK is not correct, and person *E* must log the error and take corrective action as defined by the organization's policy.

If the verification patterns show that KEK is correct, person *E* imports the PIN key using the *Key_Import* verb.

6. The system is now ready to perform PIN operations using the PIN key imported in step 5.