UNIVERSITY OF CAMBRIDGE
Computer Laboratory

The Hazards of Security API Design

Mike Bond
Computer Security Group

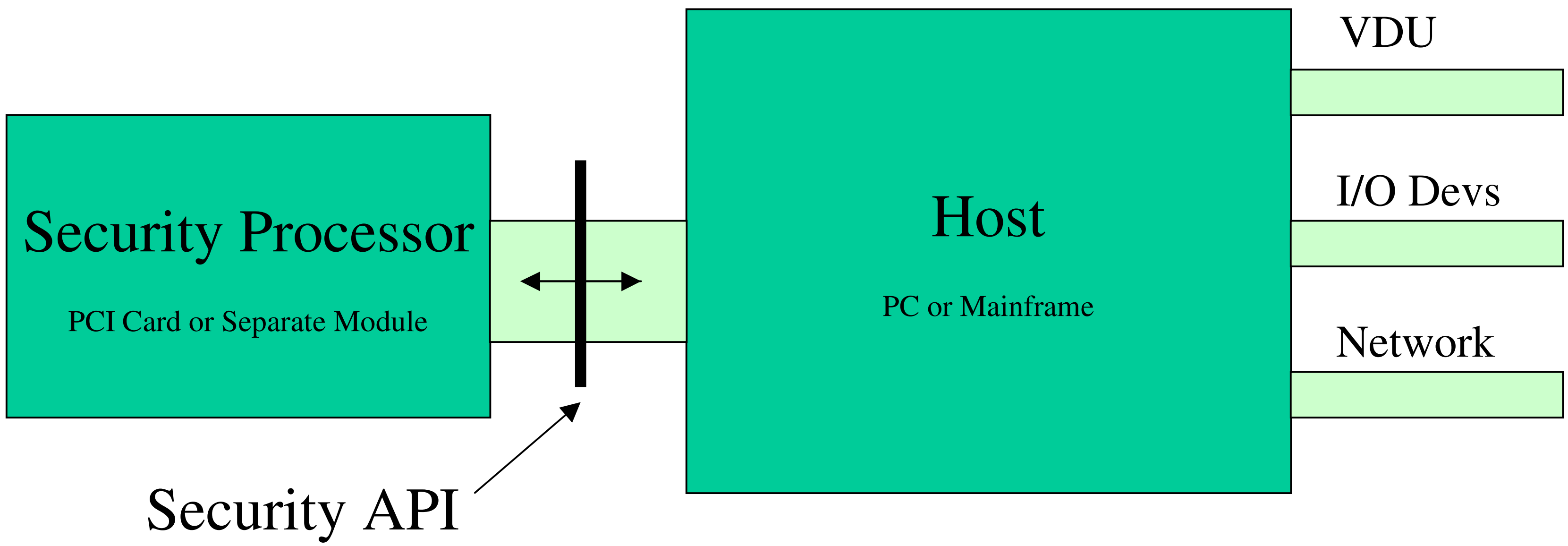
BCS Advanced Programming Group **10th January**

Talk Structure

- Introduction to Security APIs
- Case Studies
  - Visa Security Module
  - IBM 4758 CCA
  - Prism Security Module
- Summary and Conclusions

What is a Security Processor ?

- A tamper-resistant processor which uses cryptography to control processing of and access to sensitive data



Who Needs Security Processors ?

- Those who need to enforce access policies to sensitive information

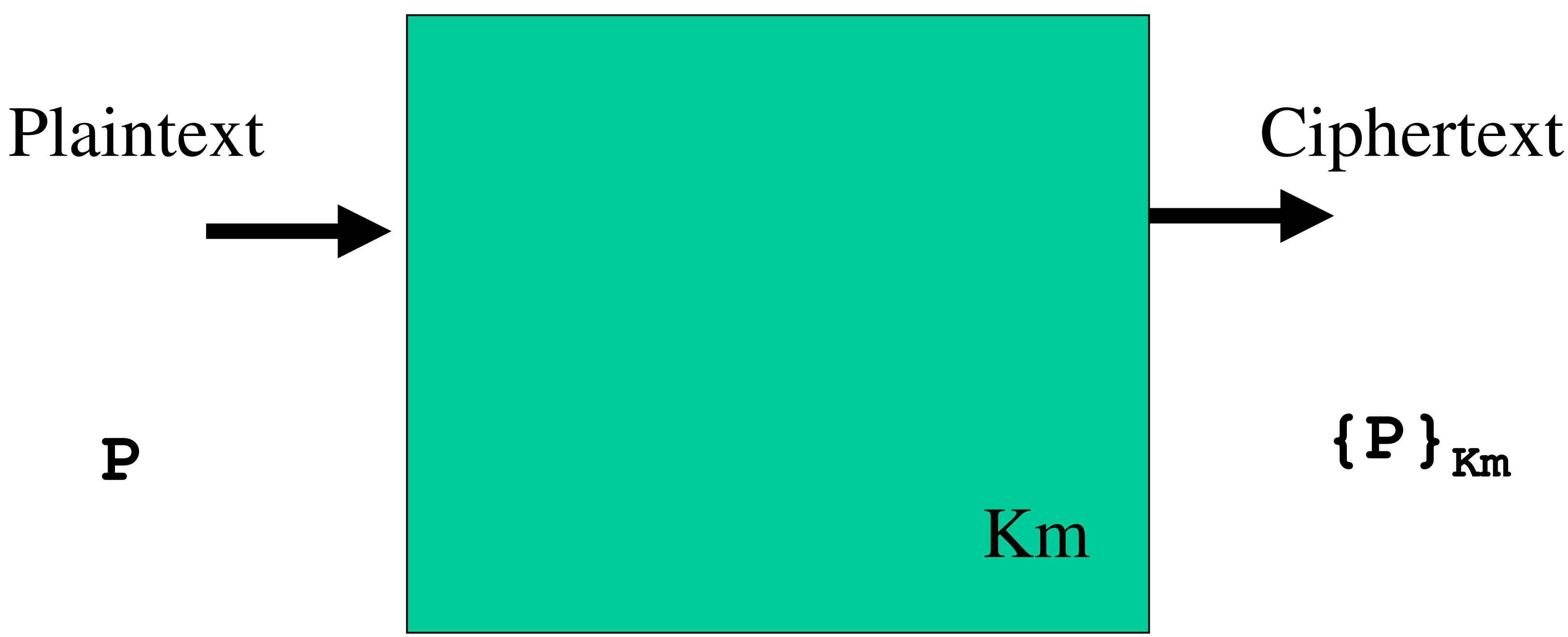
  Example: Granting signing permission at a Certification Authority
- Those who need to protect mission critical sensitive data

  Example: Protecting PIN generation keys at banks
- Those who need to protect data in hostile environments

  Example: Protecting Token Vending Machines (Electricity, National Lottery etc…)
- Those with high crypto throughput requirements

  Example: SSL acceleration for webservers

The Simplest Cryptoprocessor



Protocol Notation

- Informal notation, common in textbooks

Sender

Encryption

$$A \texttt{ -> } B : \{ X \}_{K1} , \{ KS , A , B \}_{K2}$$

Recipient

Data

Key

Concatentation

Example Security API Commands

$U \rightarrow C : \{ A \}_{KM} , \{ B \}_{KM}$

$C \rightarrow U : \{ A+B \}_{KM}$

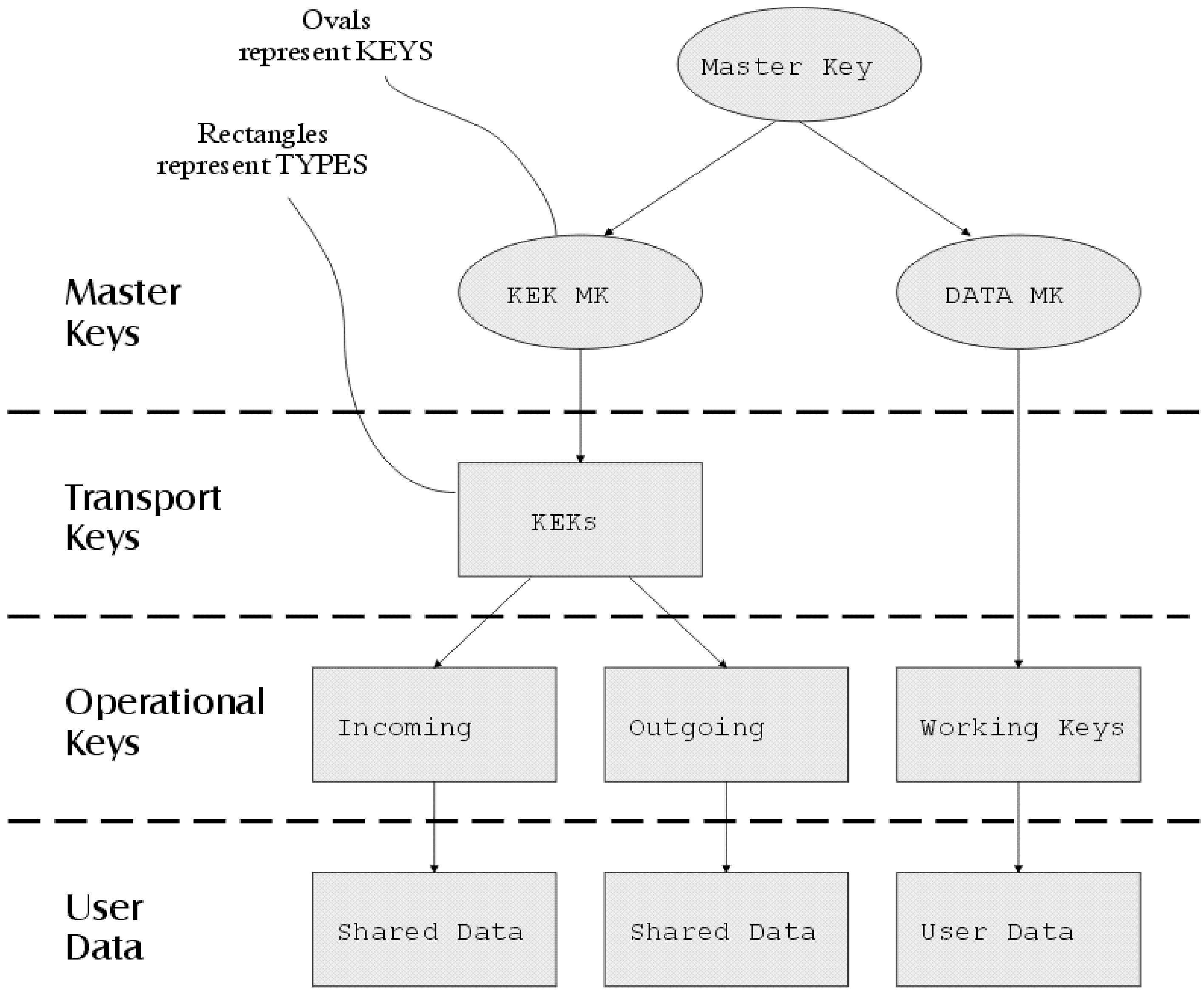
$U \rightarrow C : GUESS , \{ ANS \}_{KM}$

$C \rightarrow U :$ YES (if GUESS=ANS else NO)

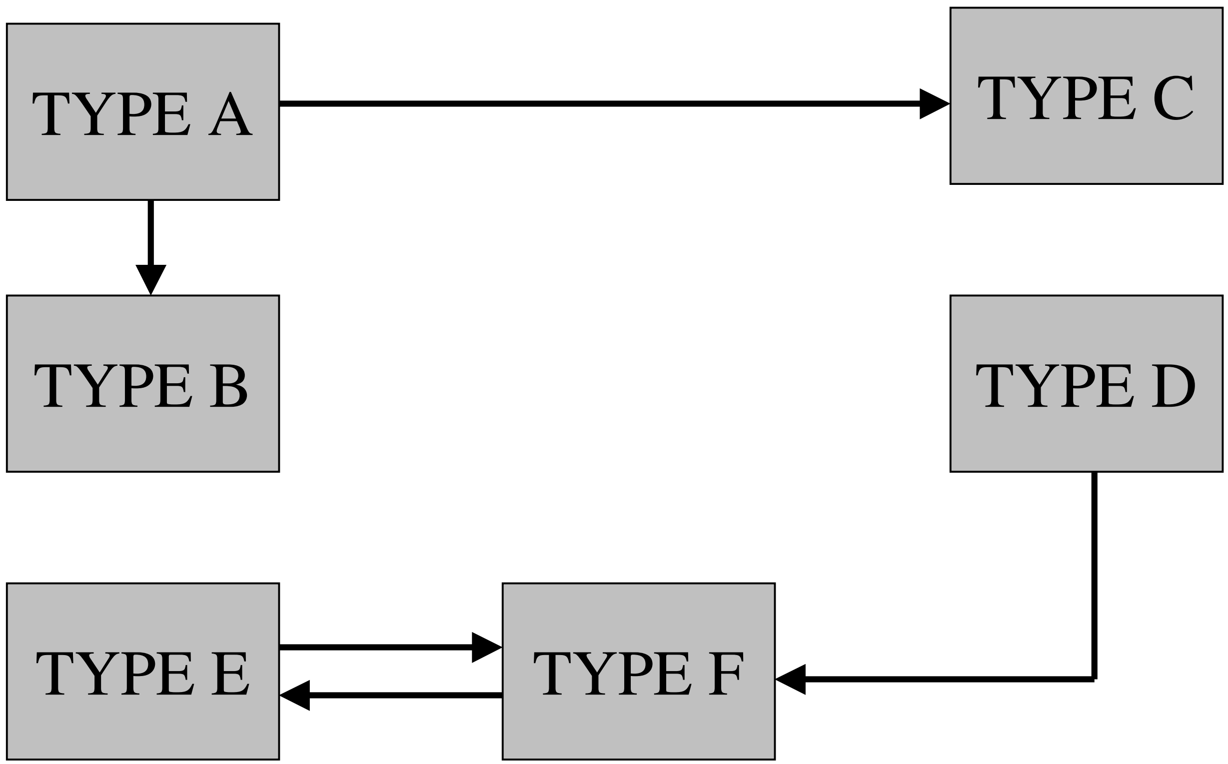
$U \rightarrow C : \{ X \}_{K1} , \{ K1 \}_{KM} , \{ K2 \}_{KM}$

$C \rightarrow U : \{ X \}_{K2}$

Example Key Hierachy

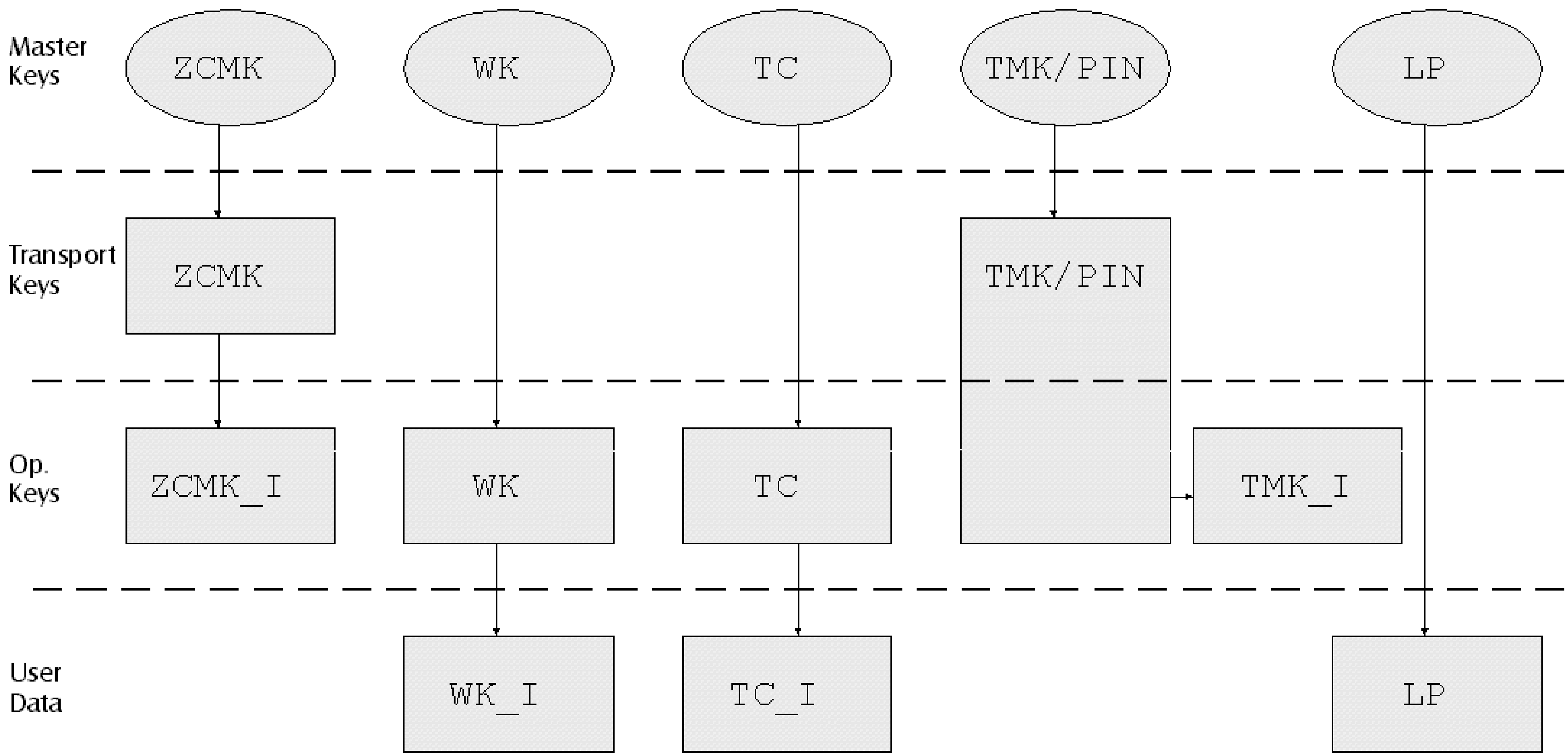


Example Type Diagram

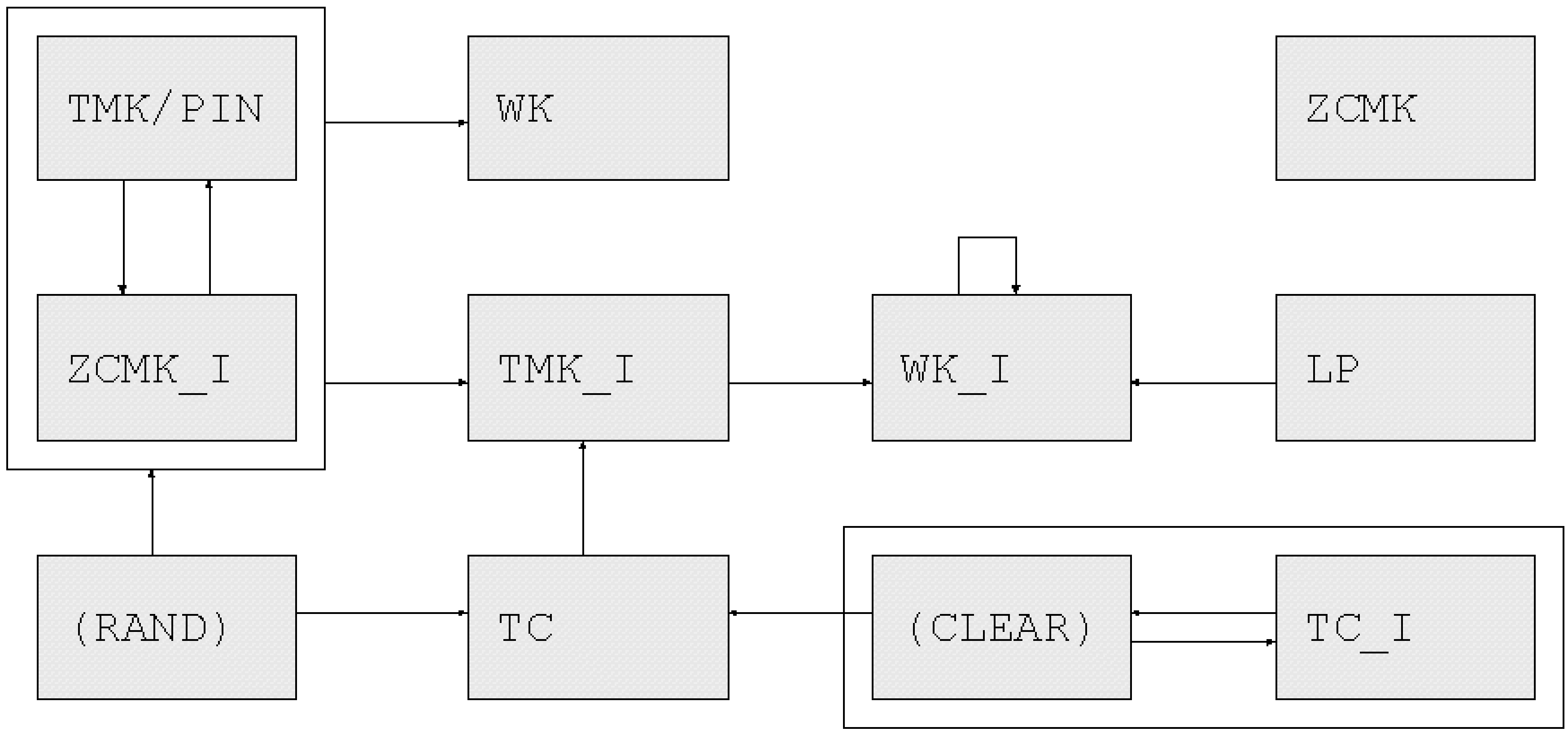


The Visa Security Module

VSM Key Hierarchy



VSM Type Diagram



What's a PIN Derivation Key ?

Start with your bank account number

0000000000052218

Encrypt with **PIN derivation key**

22BD4677F1FF34AC

Chop off the (B->1)

End 2213 (D->3)

Null Key Attack

- Top-level crypto keys exchanged between banks in several parts carried by separate couriers, which are recombined using the exclusive-OR function
- A single operator could feed in the same part twice, which cancels out to produce an ‘all zeroes’ test key. PINs could be extracted in the clear using this key

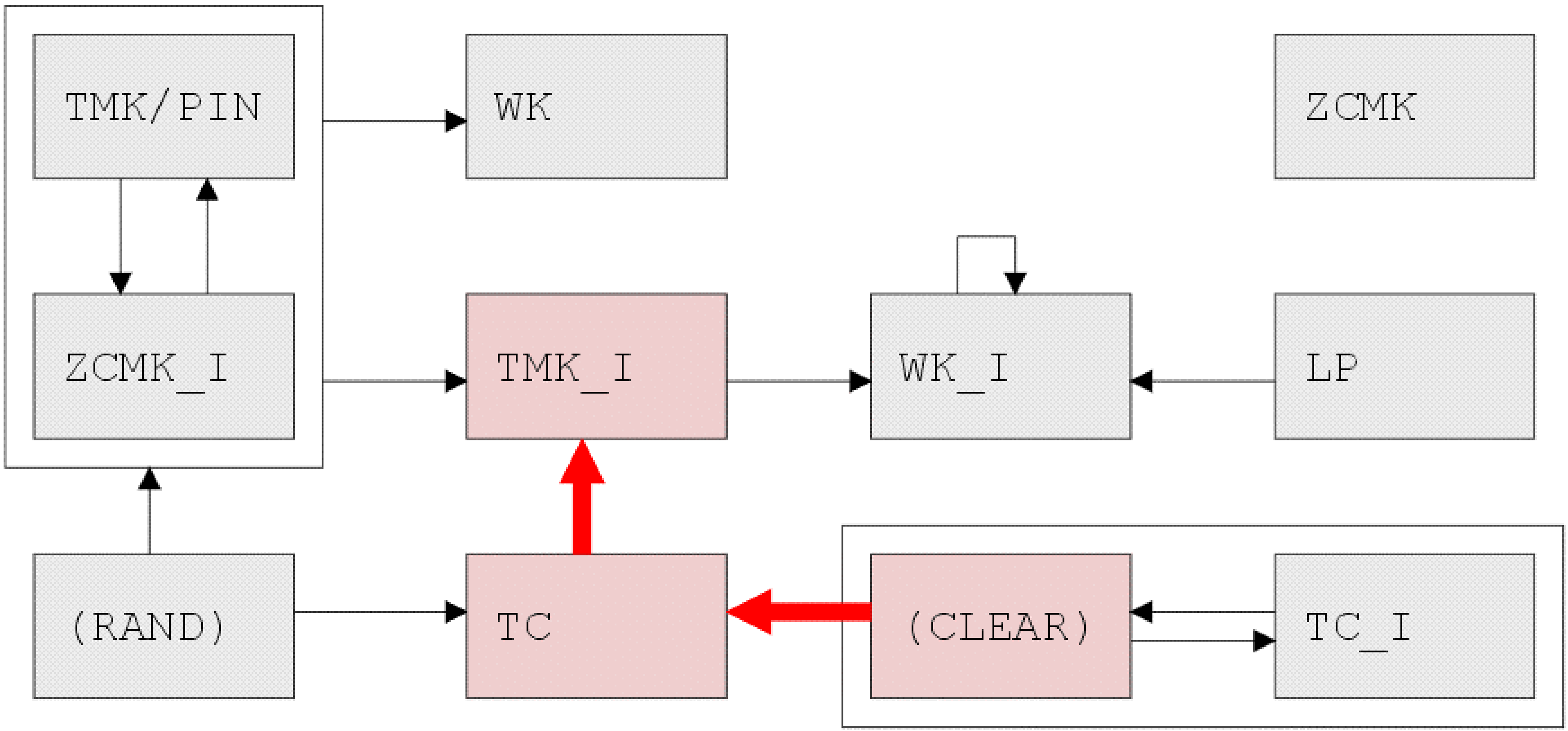
Offset Calculation Attack

- Bank adds a new command to the API to calculate the offset between a new generated PIN and the customer’s chosen PIN
- Possessing a bank account gives knowledge of one generated PIN. Any customer PIN could be revealed by calculating the offset between it and the known PIN

Type System Attack

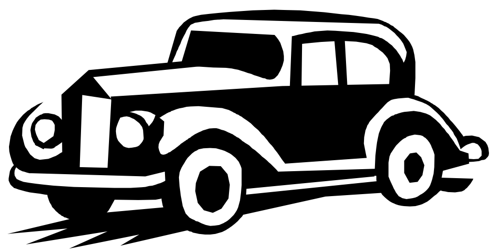
- Encrypting communication keys for transfer to an ATMs used exactly the same process as calculating a customer PIN
- Customer PINs could be generated by re-labelling an account number as a communications key, and using the same encryption process

Type System Attack



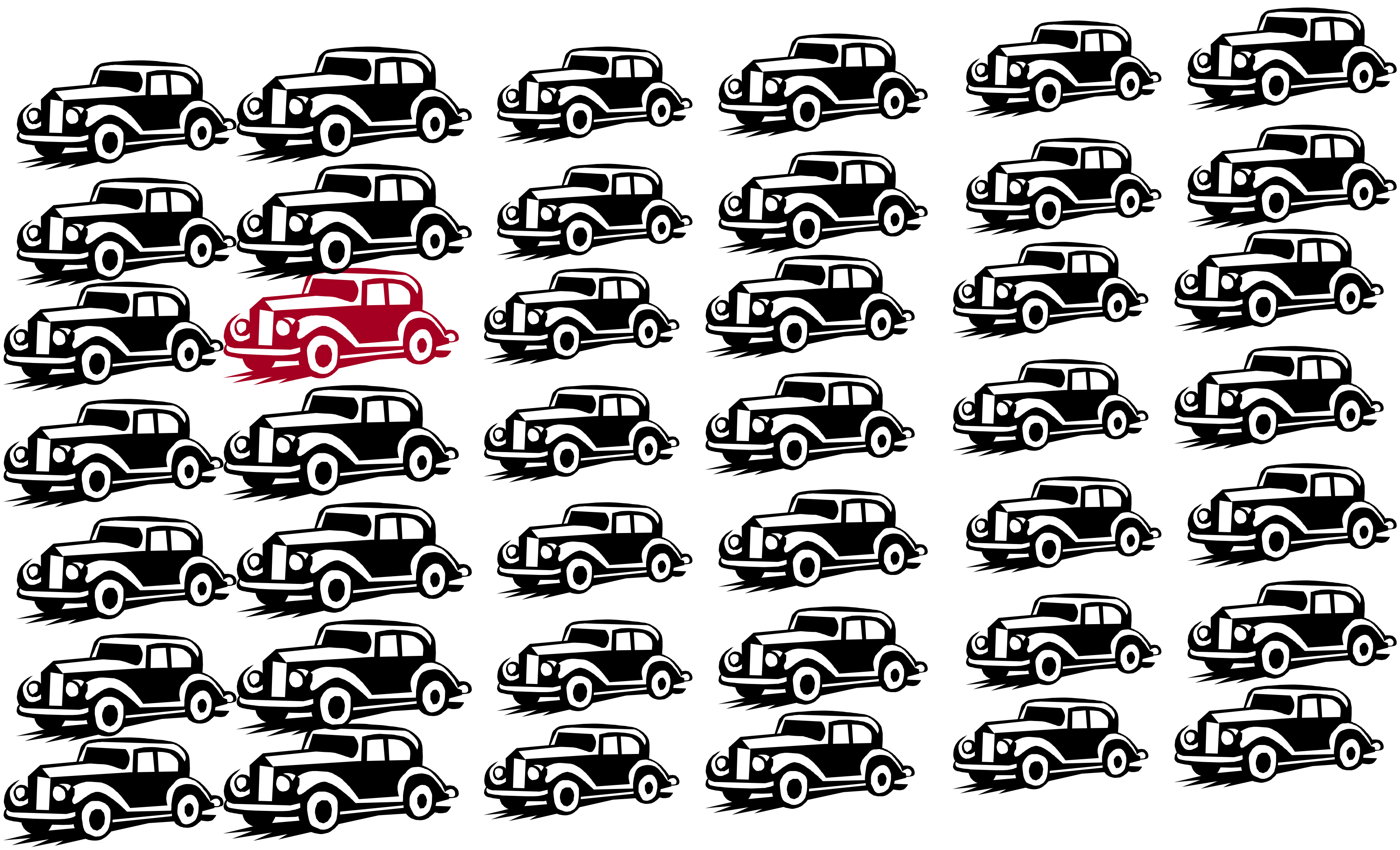
Car Park Analogy

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



- How many keys must he try?

Car Park Analogy

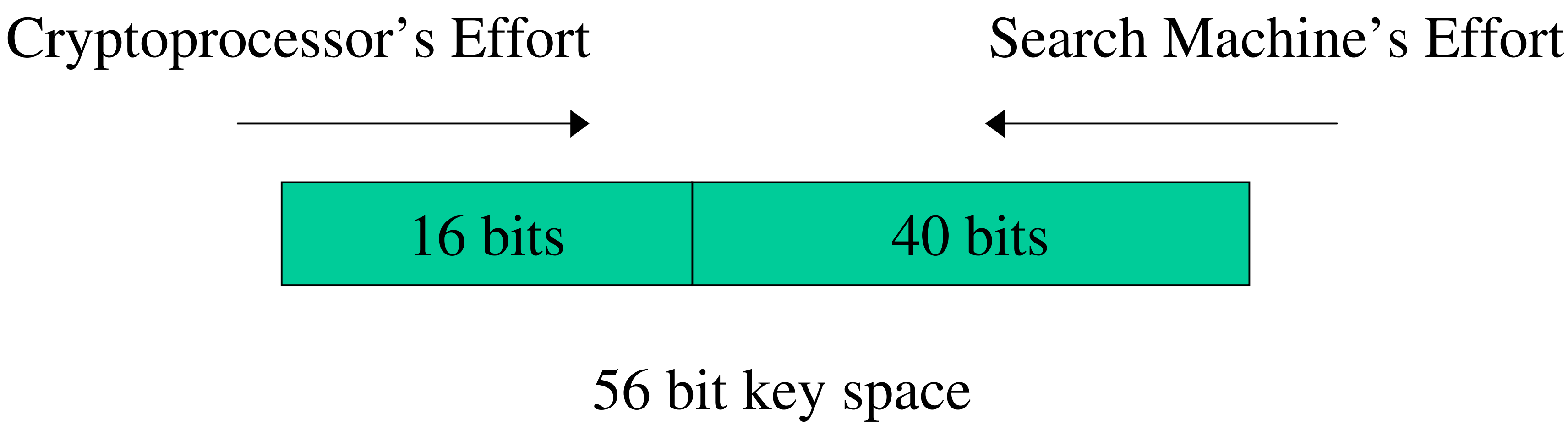


The Meet in the Middle Attack

- Common sense statistics
- Attack multiple keys in parallel
- Need the same plaintext under each key
- Encrypt this plaintext to get a 'test vector'
- Typical case: A 2^{56} search for one key becomes a 2^{40} search for 2^{16} keys

VSM MIM Attack

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



The IBM 4758

4758 Physical Protection

- Potted in epoxy resin
- Protective tamper-sensing membrane, chemically identical to potting compound
- Detectors for temperature & X-Rays
- “Tempest” shielding for RF emission
- Low pass filters on power supply rails
- Multi-stage “latching” boot sequence

= STATE OF THE ART PROTECTION!

4758 CCA Software

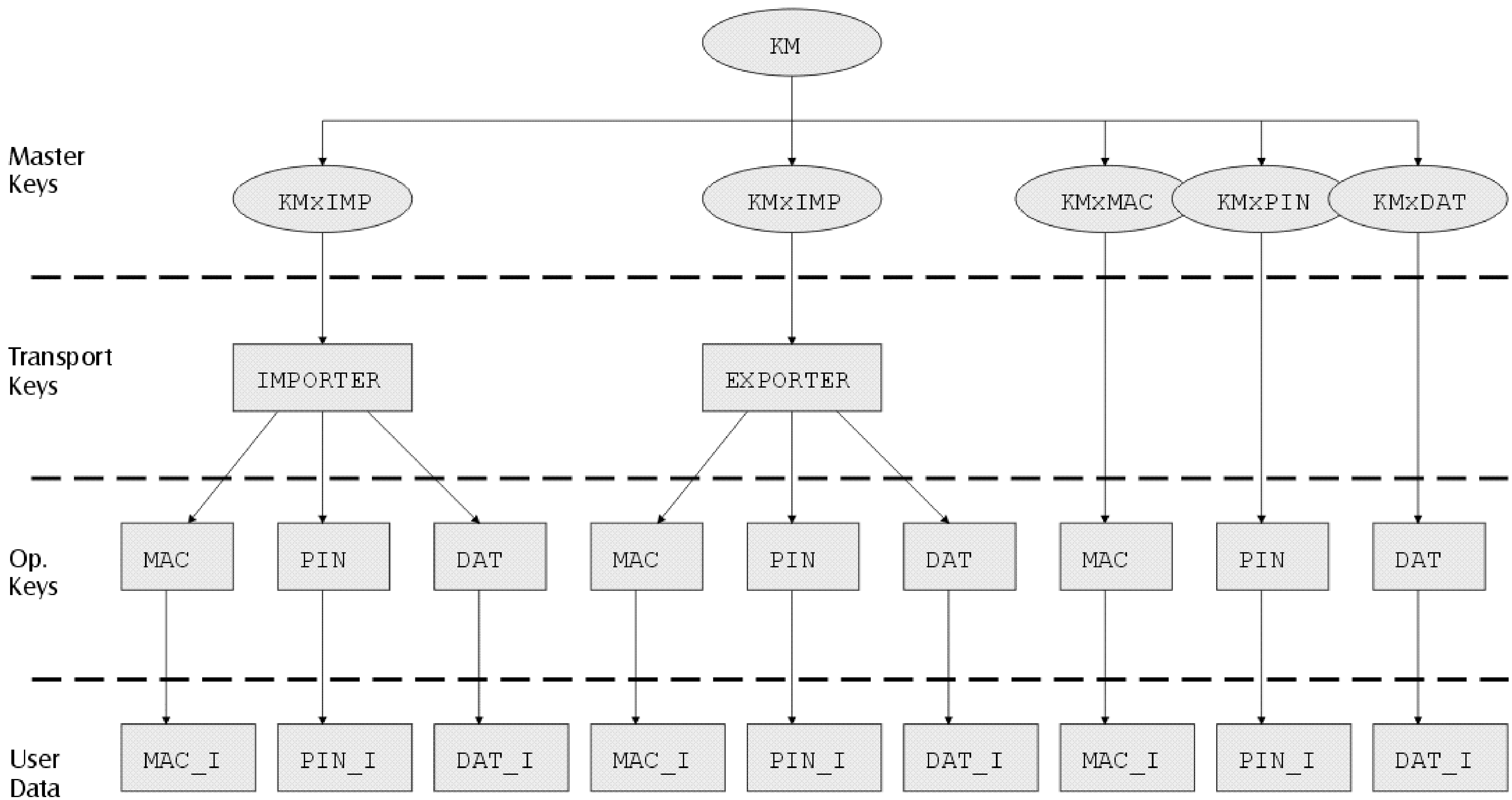
- IBM's main financial cryptography product
- In service since 1970's
- Used by PCs, Mainframes, ATMs …
- Available for NT/2000, OS/2 , AIX …
- Large and complex: roughly 150 commands, plus parameter space

Control Vectors

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

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

4758 Key Hierarchy



Key Part Import

- Thee key-part holders, each have KPA, KPC, KPC
- Final key $\mathbf{K}$ 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

$$KEK1 = KORIG$$

$$KEK2 = KORIG \oplus (old\_CV \oplus new\_CV)$$

Normally ...

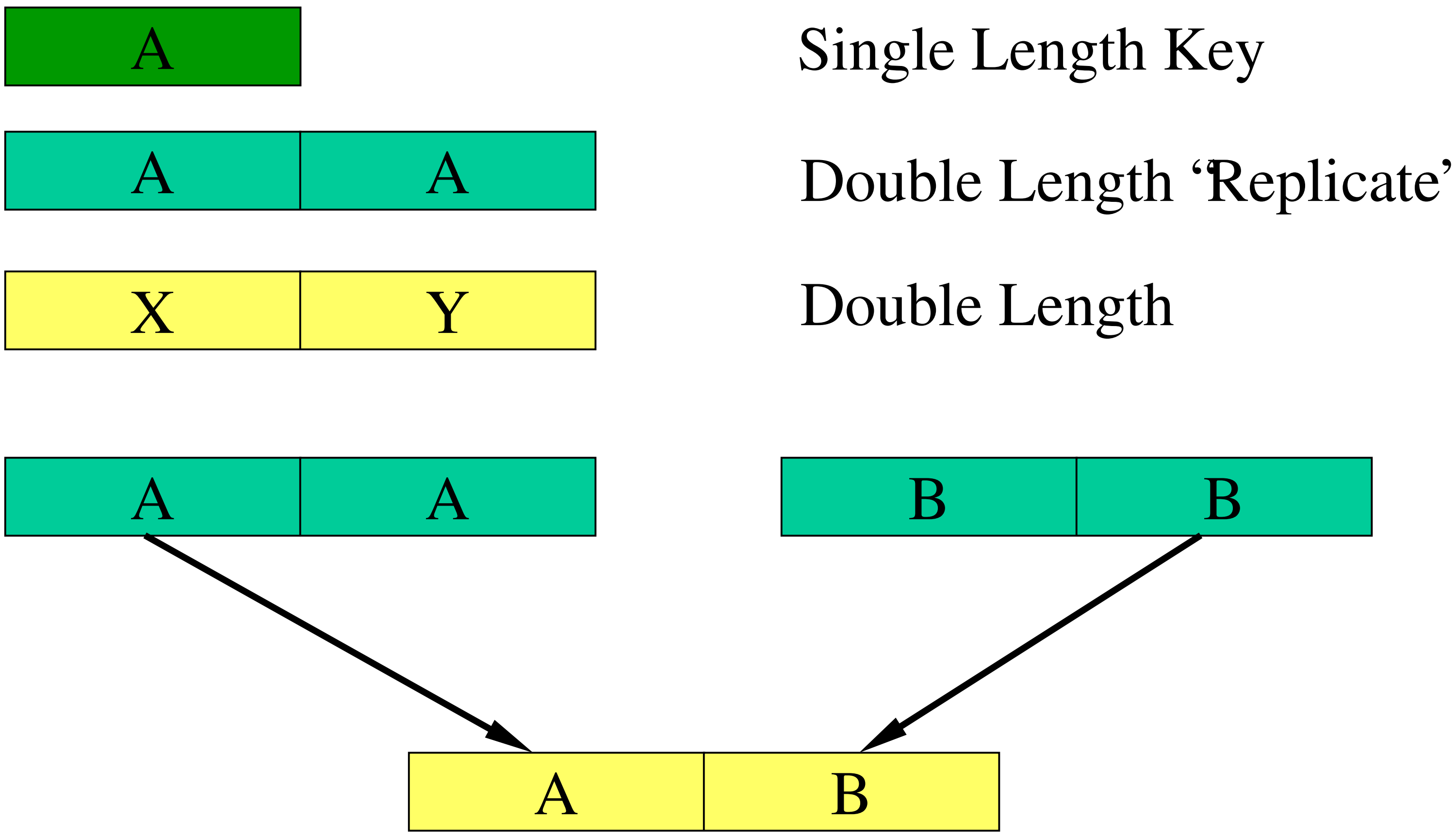
$$D_{KEK1 \oplus old\_CV}(E_{KEK1 \oplus old\_CV}(KEY)) = KEY$$

Attack ...

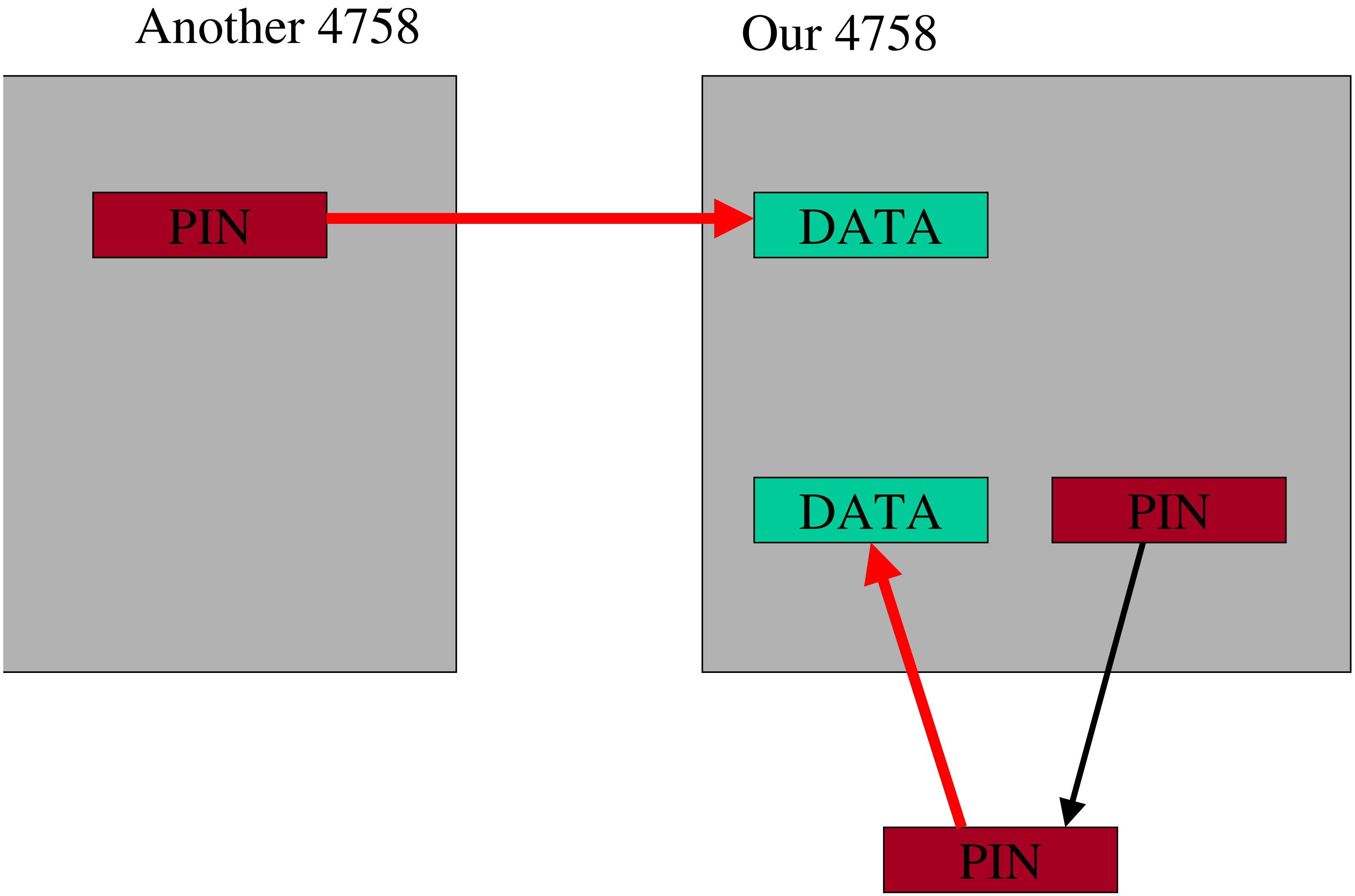
$$D_{KEK2 \oplus new\_CV}(E_{KEK1 \oplus old\_CV}(KEY)) = KEY$$

4758 Key Binding Attack

$$E_K(D_K(E_K(\ KEY\ )\ =\ E_K(KEY)$$



4758 I/E Loop Attack



Sample Code

```
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;
}
```

Publicity for 4758 CCA Attacks

- IBM initially feigned interest in attacks, and ignored repeated enquiries in first six months
- We prepared a full implementation of the attack, including special hardware to prove that it was **practical**, not just theoretical
- We warned IBM, then publicised the attack on Newsnight and in FT on $8^{th}/9^{th}$ November
- Result: international publicity, 2 x television, 5 x radio, press in UK & USA. Reuters gave internet coverage in most languages…
- Website gets ~400 hits from within ibm.com within 48 hrs, They give me a beta version of the patch by December

to be continued…

The PRISM Security Module

Prism Real-Life Application

- 2 million South African pre-payment electricity meters credited not with coins but with magic numbers bought from vending machines at local shops
- Vending machines use Prism security module to protect vending keys from shop owners/burgalars
- Discovering a vending key allows unlimited token manufacture = free electricity
- Vending keys stored in a hierarchy, with manually loaded master key at top

Master Key Entry

When vending machine first initialised…

- Three "trusted" security officers arrive with key
- Master key `Km` is a two-key triple DES key
- Each half loaded in three parts, which are exclusive-ored together
- Each security officer loads one part of each key
- Check digits returned after each load

$$\texttt{Check Digits} = \{\ 0\ \}_{\texttt{Km}}$$

Example Key Entry

Security Officer 1

SM?IK 86 08F8E3983E3BDF26

SM!IK 00 916BA78B3F290101

SM?IK 87 E92F67BFEADF91D9

SM!IK 00 0D7604EBA10AC7F3

Security Officer 2 (... n)

SM?AK 86 FD29DA10029726DC

SM!AK 00 EDB2812D704CDC34

SM?AK 87 48CCA975F4B2C8A5

SM!AK 00 0B52ED2705DDF0E4

The Faults

- Check digits are given on each half of the master key, so can attack each half separately
- After master key is loaded, anyone can continue to exclusive-or in new parts to the master key
- Can make a large set of related keys; discovery of any one of these keys lets us work back to find the master key

Making the Related Key Set

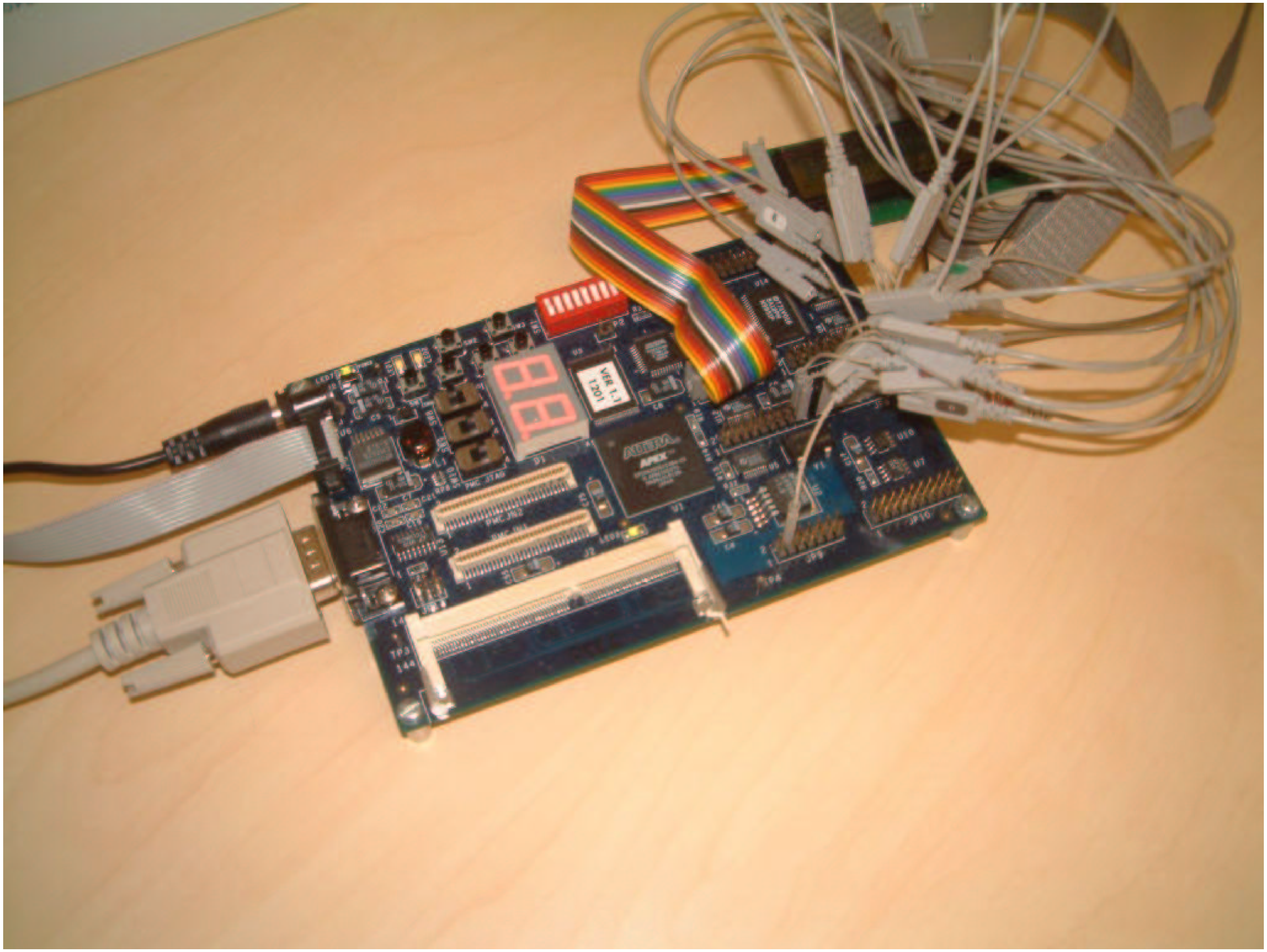
```
For I = 0000000000000001
    to  00000000001FFFF
    {
    SM?AK 87 I xor (I-1)
    SM!AK 00 (result)

    store the pair ( I , result )
    }
```

Result : 2 x ½ MB files of test vectors

Searching for a Related Key

- Used FPGA based hardware search machine
- Hardware DES implementation is ~25 times faster than the best software implementations
- Software attack with single PC would take several months
- We tried with 6 PCs (~£4500), took 3 ½ days
- Altera makes FPGA Evaluation Board with 200K gate FPGA and all software required for $995

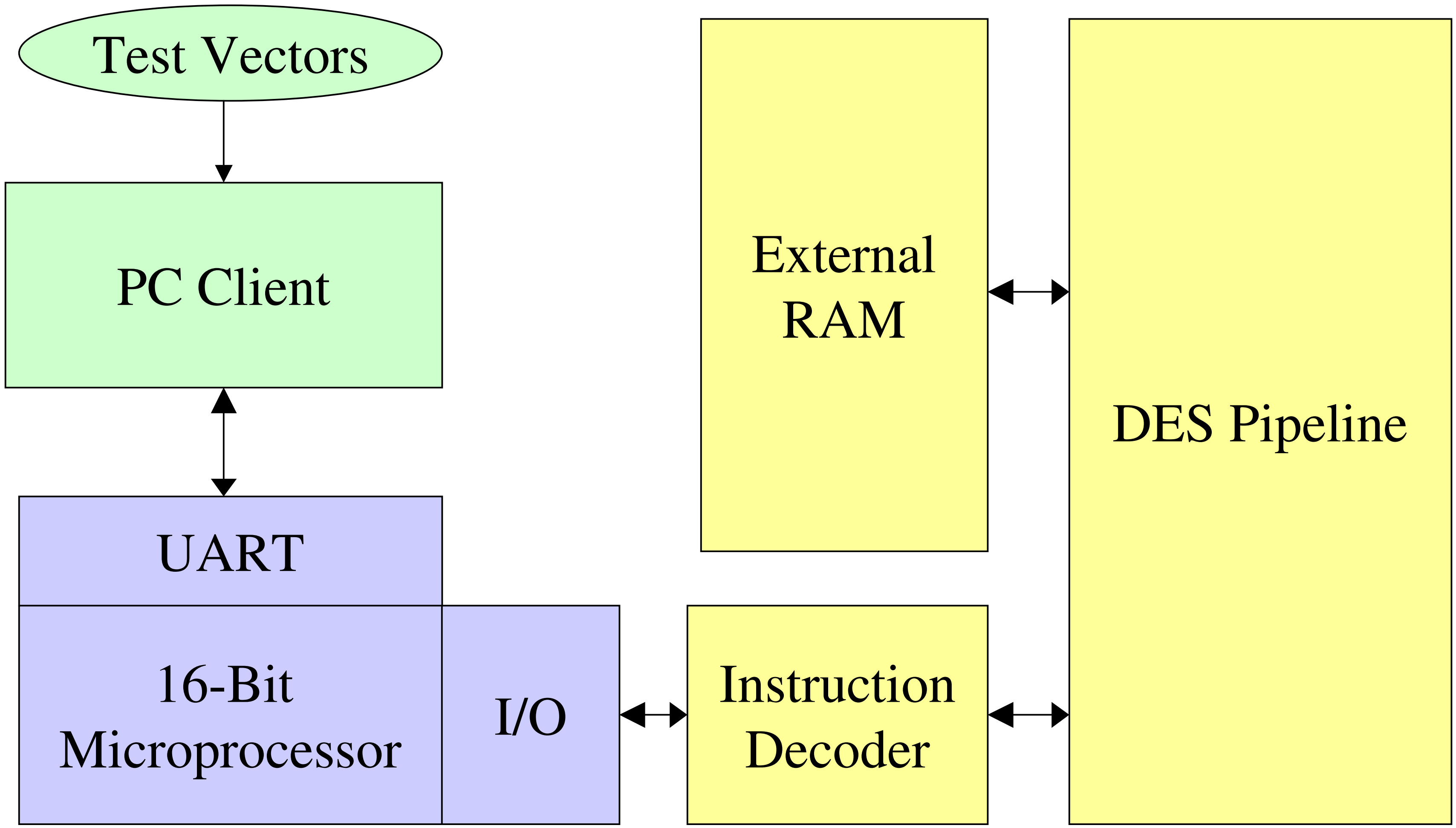
Altera Evaluation Board



Kit-based Machine

- \$1000 Excalibur kit (Altera 20K200)
  - But cost ~ \$100 for just the chip ?
- 16MHz pipeline (half speed at present)
- 2^{24} keys/second
  - 40 bit problems = 18 hours
  - 56 bit DES = 135 years (\$1M = 5..50 days)
- However.. it does 64K keys in parallel

The Big Picture



Conclusions

- Security API design is hard to get right
- Multi-purpose APIs are the hardest to get right
  - Dangerous feature interactions
  - Backwards compatibility / legacy system support is hard
- The integrity of cryptographic keys is just as important as the confidentiality
- Single DES is dead, and Triple DES must be implemented with great care
- Security API design requires a combination of protocol analysis, cryptology and threat modelling. It looks set to be a challenging and exciting research field in the future

More Information

Papers, Links & Resources

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

Attacks on IBM 4758 CCA & Hardware Cracker

http://www.cl.cam.ac.uk/~rnc1/descrack

Mike.Bond@cl.cam.ac.uk