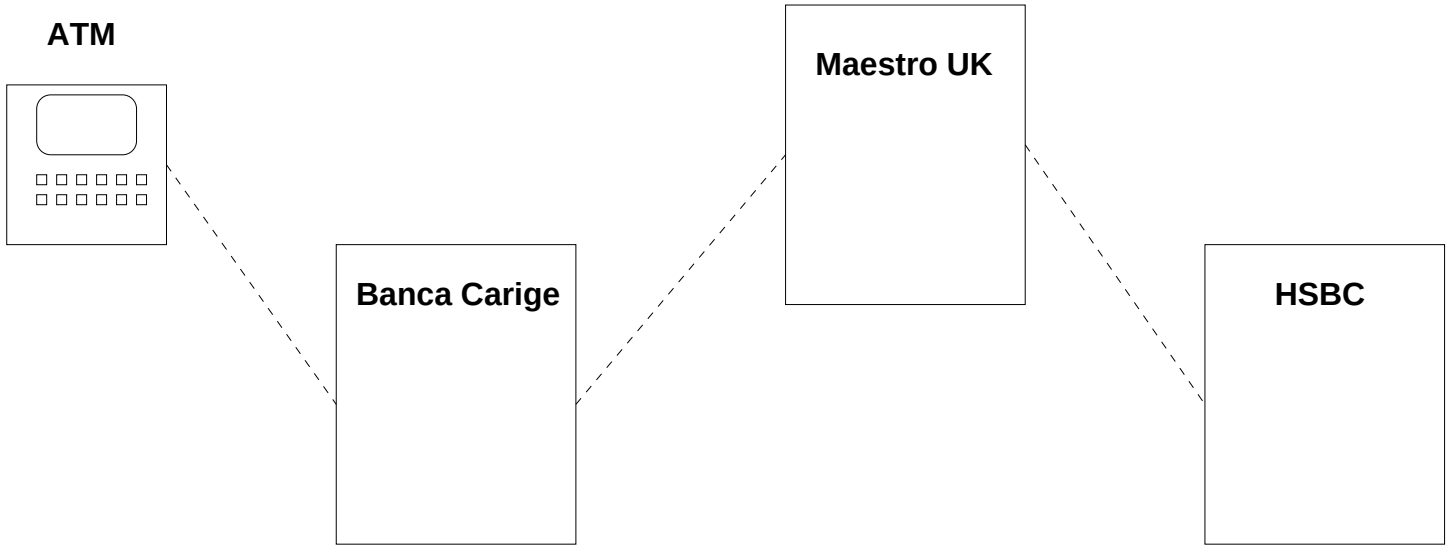
Formalised PIN Cracking

Graham Steel

School of informatics

Automated Teller Machines



Hardware Security Modules

PIN Block Attacks

ISO format 0

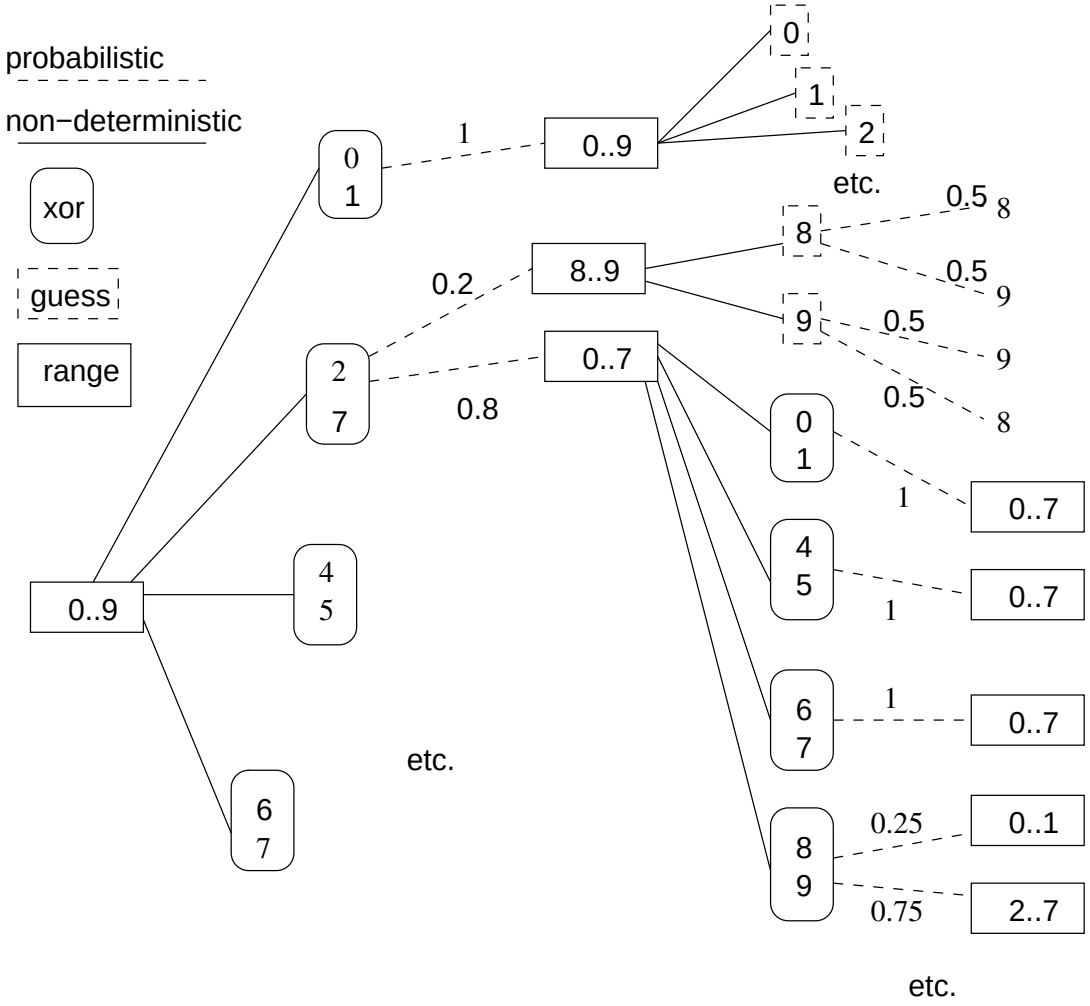
`04PPPPFFFFFFFFFF`

`0000AAAAAAAAAAAA`

VISA format 3

`PPPPFFFFFFFFFFFF`

Error check ($0 \leq P \leq 9$) leaks information


probabilistic
non−deterministic
xor
guess
range
0..9
0 1
2 7
4 5
6 7
1
0..9
0
1
2
etc.
0.2
8..9
0.8
0..7
8
9
0.5
8
0.5
9
0.5
9
0.5
8
0 1
1
0..7
4 5
1
0..7
6 7
1
0..7
8 9
0.25
0..1
0.75
2..7
etc.
etc.

Optimising the Attack

Generate full tree of possible attacks from command specs.

e.g. `command(xor_in_E_F,(6..9),(0..5)).`

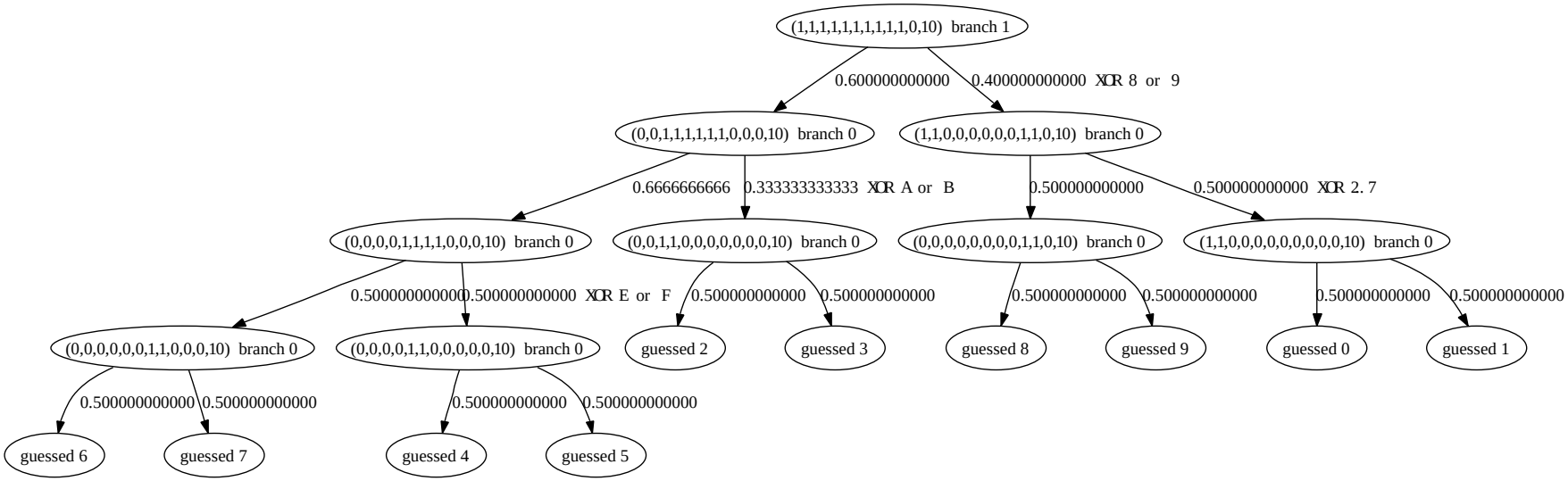
Apply PRISM - probabilistic model checker (Birmingham)

Get minimum expected number of steps to determine PIN digit: 3.4

Generate tree for best attack

School of informatics

Optimised Attack



A Word About Your PIN

Original PIN (IPIN) derived by:

3DES encrypting 0000AAAAAAAAAAAA

(The As are your PAN digits)

Decimalise result

0123456789ABCDEF

0123456789012345

PIN = IPIN + Offset (modulo 10 each digit)

Offset NOT secure!

IBM 4758 - Control Vectors

Mechanism to support many types of key: ‘role based access’

Keys stored outside box encrypted under master key XOR control vector

E.g. data keys

$\{\!|\ \text{d1}\ |\!\}_{\text{km}\oplus\text{data}}$

Encrypt Data:

$$\text{Host} \rightarrow \text{HSM} : \{\!|\ \text{d1}\ |\!\}_{\text{km}\oplus\text{data}}, \text{message}$$

$$\text{HSM} \rightarrow \text{Host} : \{\!|\ \text{message}\ |\!\}_{\text{d1}}$$

Importing Key Parts

‘Separation of duty’

Typically used to import a ‘key encrypting key’ (kek)

Key kek = k1 $\oplus$ k2

Host $\rightarrow$ HSM : k1, TYPE

HSM $\rightarrow$ Host : $\{\!|\ k1\ |\!\}_{km \oplus kp \oplus TYPE}$

Host $\rightarrow$ HSM : $\{\!|\ k1\ |\!\}_{km \oplus kp \oplus TYPE}$, k2, TYPE

HSM $\rightarrow$ Host : $\{\!|\ k1 \oplus k2\ |\!\}_{km \oplus TYPE}$

Importing Encrypted Keys

Exported from another 4758 under $KEK \oplus TYPE$

First import KEK, obtaining $\{\!|\ \text{KEK}\ |\!\}_{\text{km}\oplus\text{imp}}$

Host $\rightarrow$ HSM : $\{\!|\ \text{KEY1}\ |\!\}_{\text{KEK}\oplus\text{TYPE}}$, TYPE, $\{\!|\ \text{KEK}\ |\!\}_{\text{km}\oplus\text{imp}}$

HSM $\rightarrow$ Host : $\{\!|\ \text{KEY1}\ |\!\}_{\text{km}\oplus\text{TYPE}}$

Attack (Bond, 2001)

PIN derivation key: $\{\!|\ \text{pdk}\ |\!\}_{\text{kek}\oplus\text{pin}}$

Have key part $\{\!|\ \text{kek}\oplus\text{k3}\ |\!\}_{\text{km}\oplus\text{imp}\oplus\text{kp}}$ for known k3

Host $\rightarrow$ HSM : $\{\!|\ \text{kek}\oplus\text{k3}\ |\!\}_{\text{km}\oplus\text{kp}\oplus\text{imp}}, \text{k3}\oplus\text{pin}\oplus\text{data}, \text{imp}$

HSM $\rightarrow$ Host : $\{\!|\ \text{kek}\oplus\text{pin}\oplus\text{data}\ |\!\}_{\text{km}\oplus\text{imp}}$

Attack (Bond, 2001) (part 2)

Key Import

Host $\rightarrow$ HSM : $\{\!|\ \text{pdk}\ |\!\}_{\text{kek}\oplus\text{pin}}$, data, $\{\!|\ \text{kek}\oplus\text{pin}\oplus\text{data}\ |\!\}_{\text{km}\oplus\text{imp}}$

HSM $\rightarrow$ Host : $\{\!|\ \text{pdk}\ |\!\}_{\text{km}\oplus\text{data}}$

Encrypt data

Host $\rightarrow$ HSM : $\{\!|\ \text{pdk}\ |\!\}_{\text{km}\oplus\text{data}}$, pan

HSM $\rightarrow$ Host : $\{\!|\ \text{pan}\ |\!\}_{\text{pdk}}$ (= PIN!)

Formal Modelling

HSMs are ‘stateless’

$P(x)$ if x is ‘public’ - i.e. outside HSM

One clause for each command

Host $\rightarrow$ HSM : $\{\!|$ d1 $|\!\}_{\mathrm{km} \oplus \mathrm{data}}$, message

HSM $\rightarrow$ Host : $\{\!|$ message $|\!\}_{\mathrm{d1}}$

$$P(Msg) \wedge P(crypt(km \oplus data, D1)) \Rightarrow P(crypt(D1, Msg))$$

The Problem with XOR

$P(x) \wedge P(y) \rightarrow P(x \oplus y)$

Associativity and Commutativity

Self-Inverse ($a \oplus b \oplus a \equiv b$)

XOR constraints

Host $\rightarrow$ HSM : $\{\!|\ \text{KEY1}\ |\!\}_{\text{KEK}\oplus\text{TYPE}}$, TYPE, $\{\!|\ \text{KEK}\ |\!\}_{\text{km}\oplus\text{imp}}$

HSM $\rightarrow$ Host : $\{\!|\ \text{KEY1}\ |\!\}_{\text{km}\oplus\text{TYPE}}$

$$P(crypt(X,Key)) \wedge P(Type) \wedge P(crypt(km \oplus imp, Kek))$$
$$\Rightarrow P(crypt(km \oplus Type, decrypt(Kek \oplus Type, crypt(X,Key)))).$$
$$\Rightarrow decrypt(K, crypt(K,X)) = X.$$

$$P(crypt(X,Key)) \wedge P(Type) \wedge P(crypt(km \oplus imp, Kek))$$
$$\Rightarrow P(crypt(km \oplus Type, Key)) \quad IF \quad Kek \oplus Type =_{xor} X.$$

Checking Solubility

Permit only inferences which leave soluble constraints

Check:

- If there are any variables at XOR positions, it is soluble
- Otherwise count up all terms. If there are an even number of each term, it is soluble. If not, insoluble.

Store in normal form

$$x_1 \oplus \ldots \oplus x_n \quad = \quad t_1 \oplus \ldots \oplus t_n$$

Subsumption Checking

If C_1 subsumes C_2 without consideration of XOR constraints, then it is a valid subsumer iff:

1. C_1 has no XOR constraint

or

2. C_1 and C_2 have the same XOR constraints after substitutions applied

Results

Implemented in daTac, [Vigneron, 1994]

- Bond's attack shown above
- Import/Export Attack (also due to Bond)
- IBM's own attack
- Attack on NSPKL variant - Jacquemard et al. model

PIN Decimalisation Table

Standard

```
0123456789ABCDEF
0123456789012345
```

Attack

```
0123456789ABCDEF
1123456789012345
```

Alter offset to establish effect of change

Further Work

- PIN Block format analysis
- Improvement to XOR constraint solving
- Ideas for decimalisation table attacks

Summary

- API analysis exciting new area!

  Used also in smartcards, POS devices, mobile phones, DRM, ...

- Some early successes, many problems remain

  XOR constraints, probabilistic model checking look good

  Need ideas for decimalisation table attacks

  There are also other kinds of attacks

---

http://dream.inf.ed.ac.uk/projects/aascs/