TROOPERS
The IT-Security Conference

Forging Canon Original Decision Data

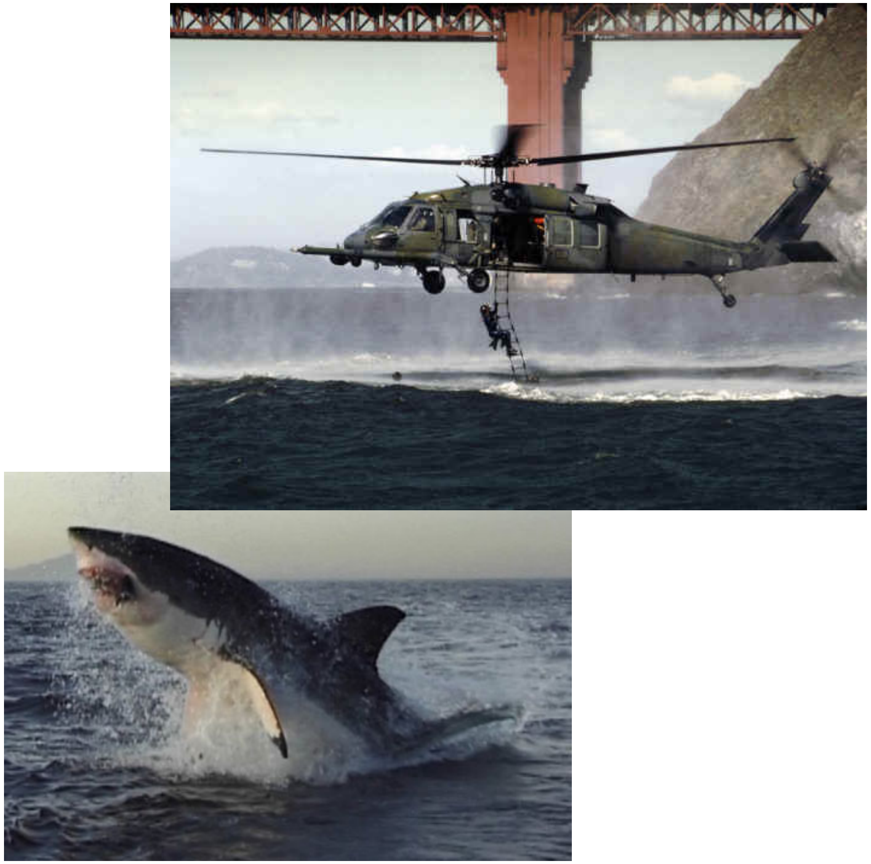
TROOPERS II
28 March – 1 April 2011
Heidelberg, Germany

Dmitry Sklyarov

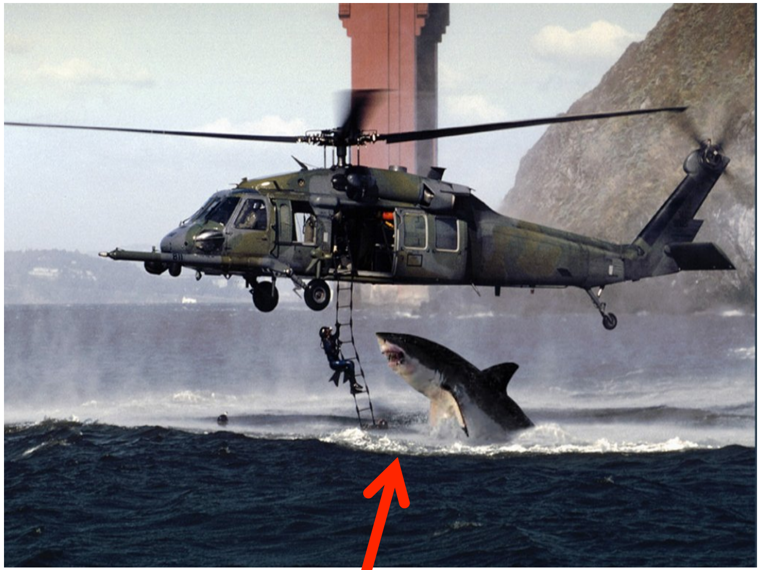
ELCOMSOFT
PROACTIVE SOFTWARE

What is Original Decision Data

ODD is added to the image file by camera and expected to provide information to detect any image alteration



It is too easy to edit photos…



Modified!

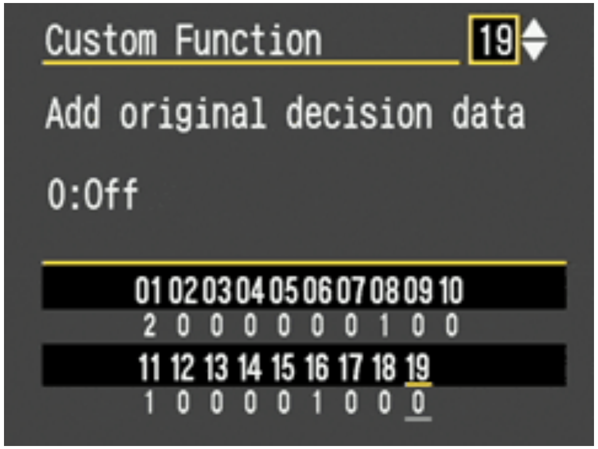
My first DSLR – Canon EOS 350D

- Great piece of hardware
- Like it very much! (honestly! :)

- Does not support Original Image Verification features :(

My next DSLR – Canon EOS 30D

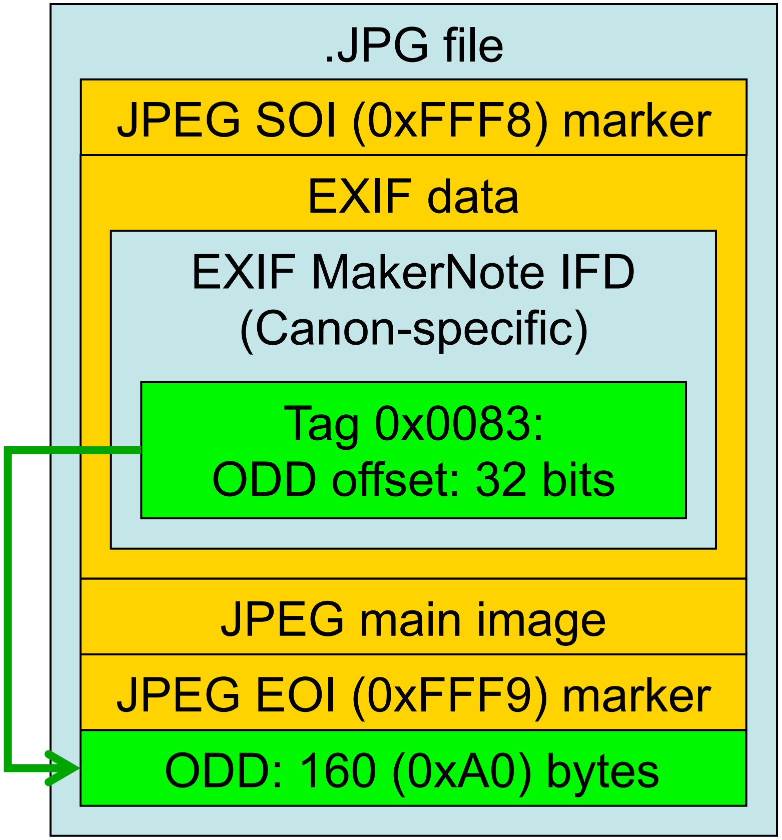
- Even better than 350D :)
- Custom Function 19: Add Original Decision Data to each picture taken



```
Custom Function                 19
Add original decision data
0:Off
01 02 03 04 05 06 07 08 09 10
 2  0  0  0  0  0  0  1  0  0
11 12 13 14 15 16 17 18 19
 1  0  0  0  0  1  0  0  0
```

EOS 30D: ODD in .JPG file

- ODD is stored as 160 (0xA0) bytes appended after JPEG EOI (0xFFD9) marker
- File offset of ODD is stored as 32-bit value in Tag 0x0083 inside EXIF/MakerNote IFD (Image File Directory)



.JPG file

JPEG SOI (0xFFF8) marker

EXIF data

EXIF MakerNote IFD
(Canon-specific)

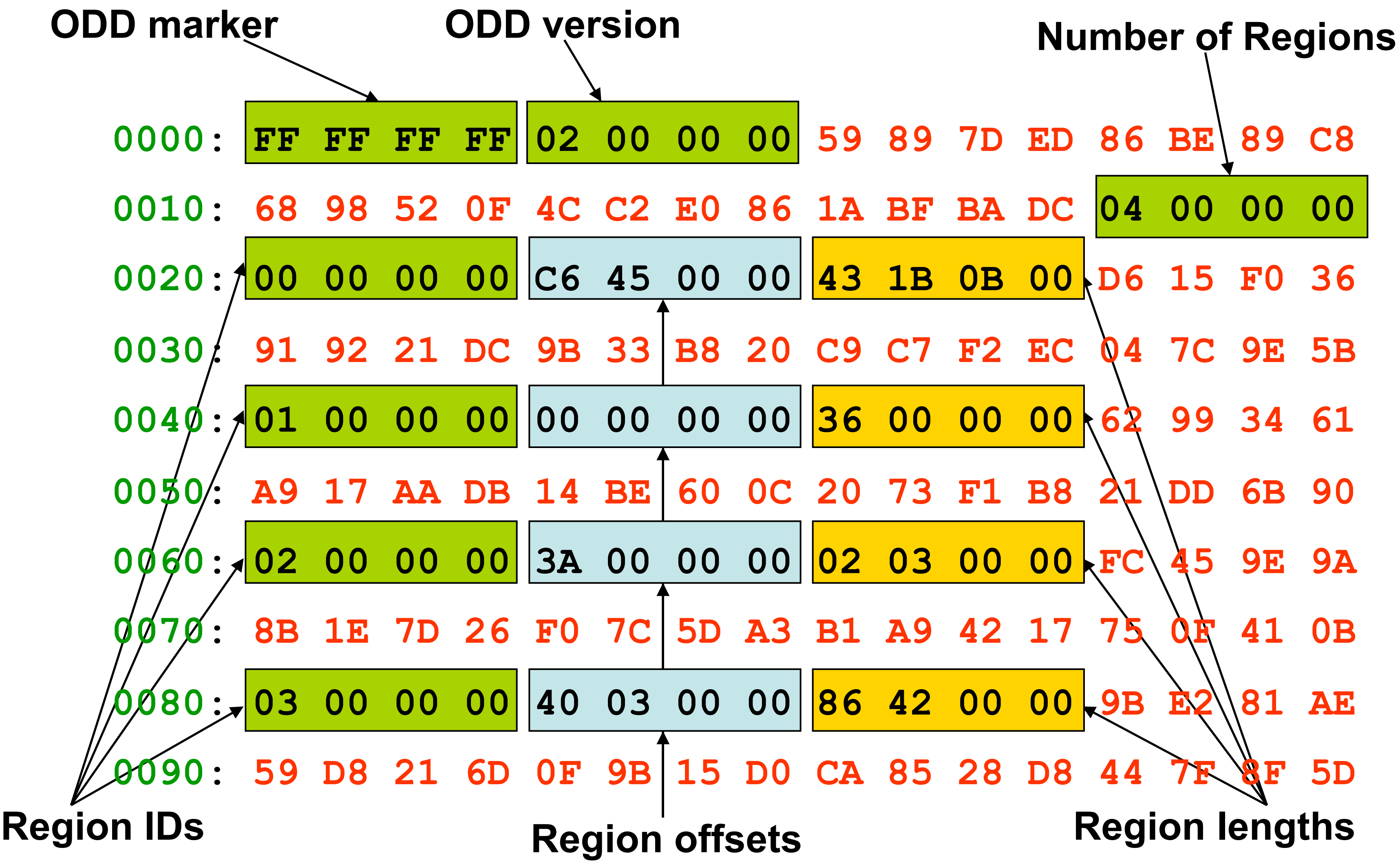
Tag 0x0083:
ODD offset: 32 bits

JPEG main image

JPEG EOI (0xFFF9) marker

ODD: 160 (0xA0) bytes

EOS 30D: ODD dump



ODD marker

ODD version

Number of Regions

```
0000: FF FF FF FF 02 00 00 00 59 89 7D ED 86 BE 89 C8
0010: 68 98 52 0F 4C C2 E0 86 1A BF BA DC 04 00 00 00
0020: 00 00 00 00 C6 45 00 00 43 1B 0B 00 D6 15 F0 36
0030: 91 92 21 DC 9B 33 B8 20 C9 C7 F2 EC 04 7C 9E 5B
0040: 01 00 00 00 00 00 00 00 36 00 00 00 62 99 34 61
0050: A9 17 AA DB 14 BE 60 0C 20 73 F1 B8 21 DD 6B 90
0060: 02 00 00 00 3A 00 00 00 02 03 00 00 FC 45 9E 9A
0070: 8B 1E 7D 26 F0 7C 5D A3 B1 A9 42 17 75 0F 41 0B
0080: 03 00 00 00 40 03 00 00 86 42 00 00 9B E2 81 AE
0090: 59 D8 21 6D 0F 9B 15 D0 CA 85 28 D8 44 7F 8F 5D
```

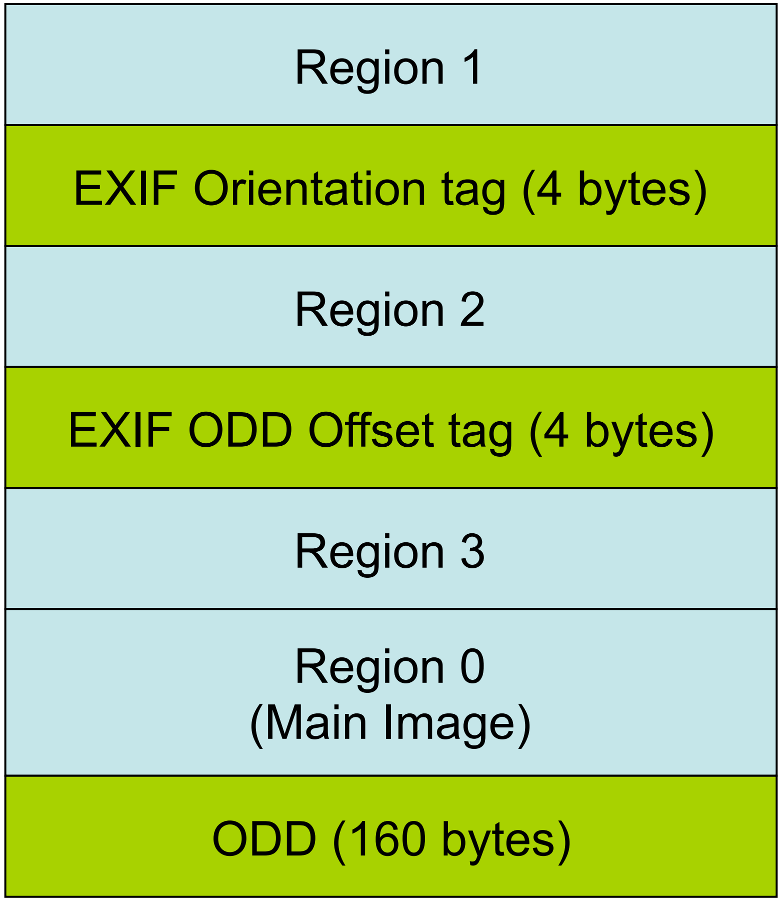
Region IDs

Region offsets

Region lengths

ODDv2: Regions layout

- R0: Main Image
- R1: From 0 to Tag 0x0112 of EXIF Main IFD (Orientation tag)
- R2: From Orientation tag to ODD Offset tag
- R3: From ODD Offset tag to Main Image



Region 1

EXIF Orientation tag (4 bytes)

Region 2

EXIF ODD Offset tag (4 bytes)

Region 3

Region 0
(Main Image)

ODD (160 bytes)

ODDv2: General structure

```
typedef struct {
  DWORD marker;             // Marker == ~0
  DWORD ver;                // ODD version == 2
  BYTE unknown_1[20];
  DWORD nRegions;           // Number of Regions == 4
  struct {
    DWORD id;               // Region ID
    DWORD o;                // Region data offset
    DWORD cb;               // Region data length
    BYTE unknown_2[20];
  } r[4];                   // Regions
} T_ODD_v2;                 // sizeof(T_ODD_v2) == 0xA0
```

ODDv2: Guessing unknowns

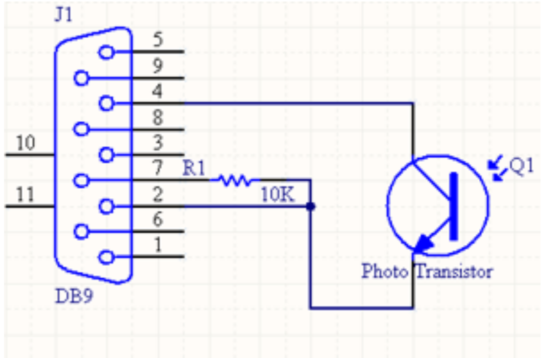
Field before regions definition	Represents signature for the whole image file?
Field inside region definition	Hold signature of the particular region data?
Signature length is always 20 bytes	Too short for asymmetric, but matches SHA-1 length
Symmetric SHA-1 based authentication?	May be HMAC-SHA-1?

Looking into camera’s firmware

Three easy steps :)

1. Explore CHDK (Canon Hacker's Development Kit) Wiki and forum

2. Dump firmware using “blinking” or some other technique



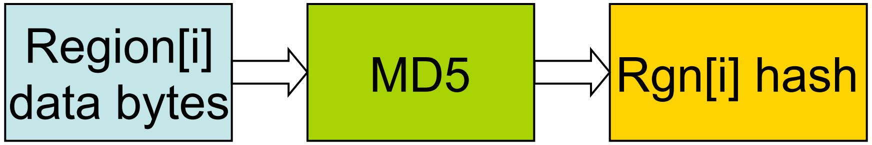
3. Use IDA Pro to analyze dumped ARM code

ODDv2: Clarified structure

```
typedef struct {
  DWORD marker;         // Marker == ~0
  DWORD ver;            // ODD version == 2
  BYTE imgHMAC[SHA_DIGEST_LENGTH];
  DWORD nRegions;       // Number of Regions == 4
  struct {
    DWORD id;           // Region ID
    DWORD o;            // Region data offset
    DWORD cb;           // Region data length
    BYTE HMAC[SHA_DIGEST_LENGTH];
  } r[4];               // Regions
} T_ODD_v2;             // sizeof(T_ODD_v2) == 0xA0
```

ODDv2: Region HMAC

- Hash region data bytes with MD5



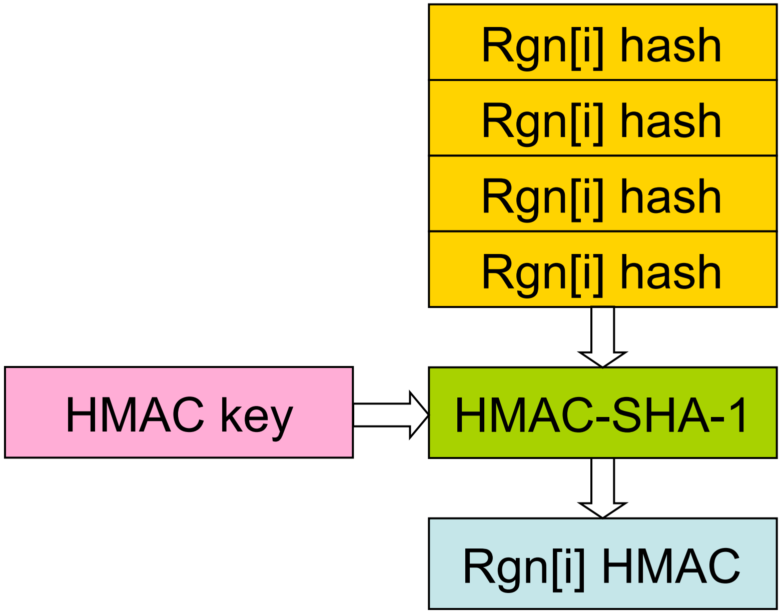
Region[i] data bytes

MD5

Rgn[i] hash

- Repeat 128-bit region hash value 4 times to fill 64-byte buffer

- Calculate HMAC for the buffer, store result in ODDv2.r[i].HMAC



Rgn[i] hash

Rgn[i] hash

Rgn[i] hash

Rgn[i] hash

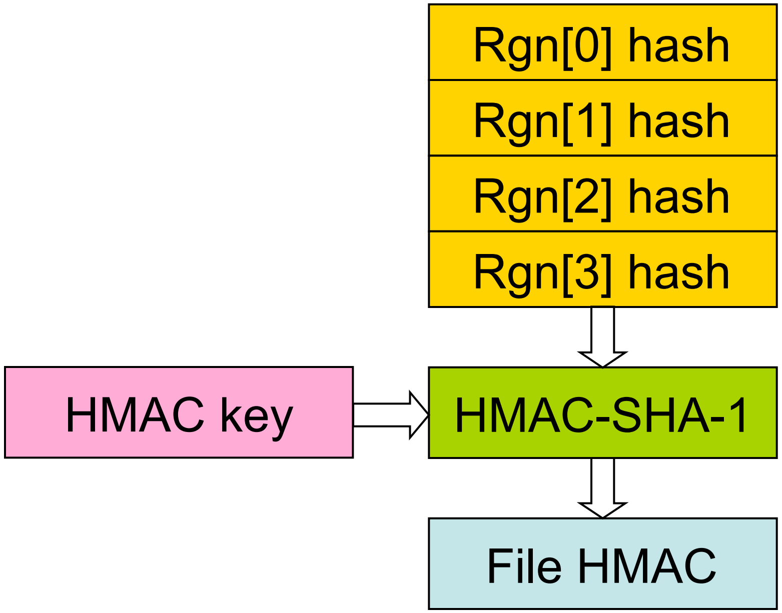
HMAC key

HMAC-SHA-1

Rgn[i] HMAC

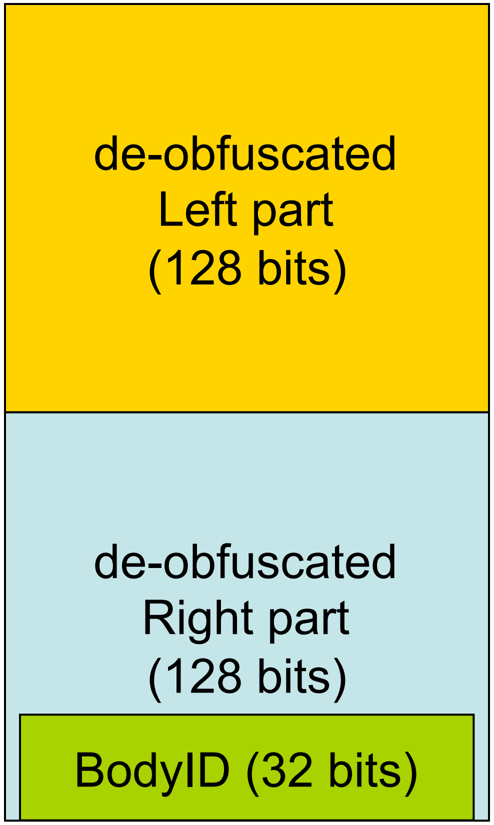
ODDv2: Image file HMAC

- Merge four 128-bit hash values for all 4 regions to fill 64-byte buffer
- Calculate HMAC for the buffer, store result in ODDv2.imgHMAC



ODDv2: What is HMAC key

- Length is 256 bits (32 bytes)
- Builds from two 128-bit parts, each part is stored separately in obfuscated form
- Last 32 bits are replaced by camera's BodyID (stored in EXIF) before HMAC calculation



ODDv2: Notes on HMAC key

- Key value is the same for all cameras of some particular model (e.g. EOS 30D)
- Different camera models (5D, 20D, 30D) uses different keys

- Knowing key for particular model allows forging ODD for any camera of that model!
- Key can be extracted from the camera!

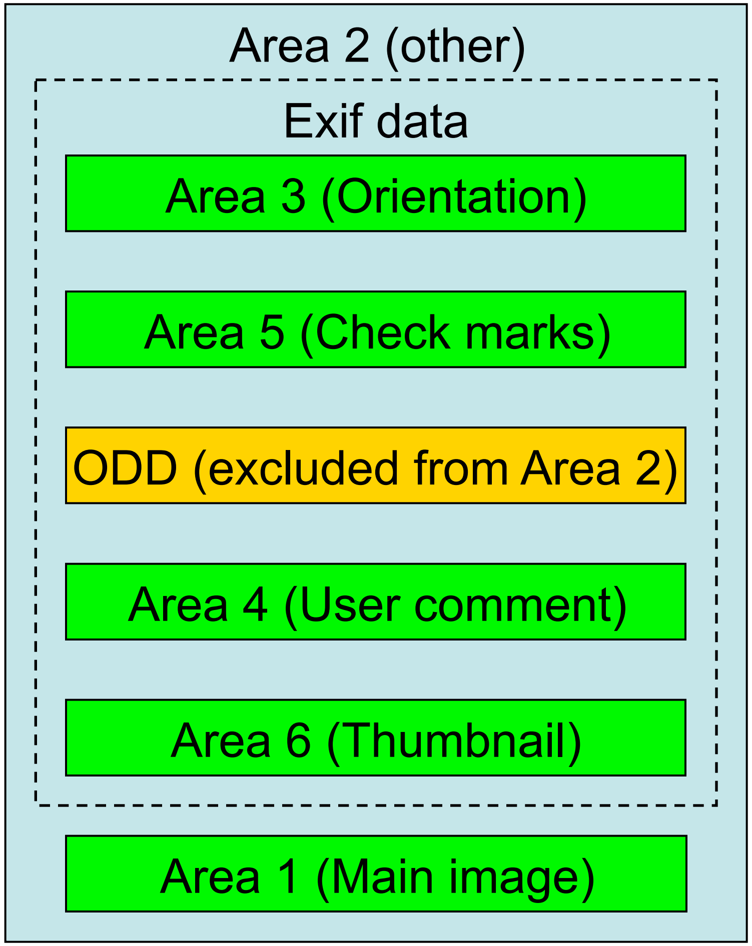
EOS 40D: New version of ODD

- ODD is stored within EXIF
- ODD version is 3
- Image file length is stored inside ODD
- File is treated as set of areas (based on content type)
- Area could contain several regions
- Integrity of each area monitored independently

ODDv3: Area layout in .JPG file

1. Main image
2. All other data
3. Orientation
4. User comment
5. Check marks
6. Thumbnail

Note: ODD is not included in any area



Area 2 (other)

Exif data

Area 3 (Orientation)

Area 5 (Check marks)

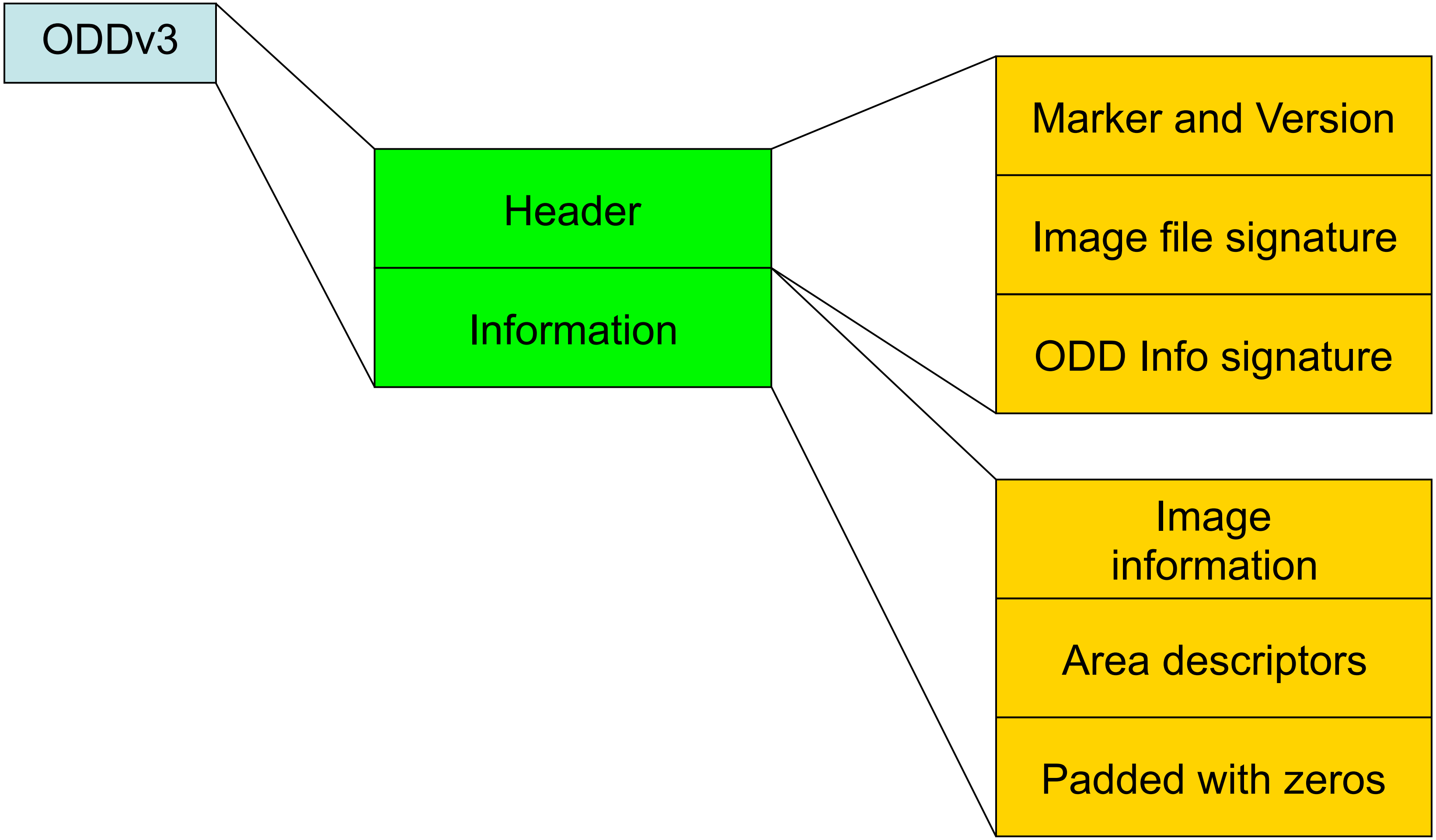
ODD (excluded from Area 2)

Area 4 (User comment)

Area 6 (Thumbnail)

Area 1 (Main image)

ODDv3: General structure



ODDv3

Header

Information

Marker and Version

Image file signature

ODD Info signature

Image information

Area descriptors

Padded with zeros

ODDv3: Header structure

```
typedef struct {
  DWORD marker;           // Marker == ~0
  DWORD ver;         // ODD version == 3
  DWORD cbImgSig;      // len(Sign(Image))
  BYTE imgSig[cbImgSig];   // Sign(Image)
  DWORD cbInfoSig;     // len(Sign(oddInfo))
  BYTE infoSig[cbInfoSig]; // Sign(oddInfo)
} T_ODDv3_Hdr;
```

- Note: `cbImgSig` and `cbInfoSig` are always == 20 == `SHA_DIGEST_LENGTH`

ODDv3: Area structure

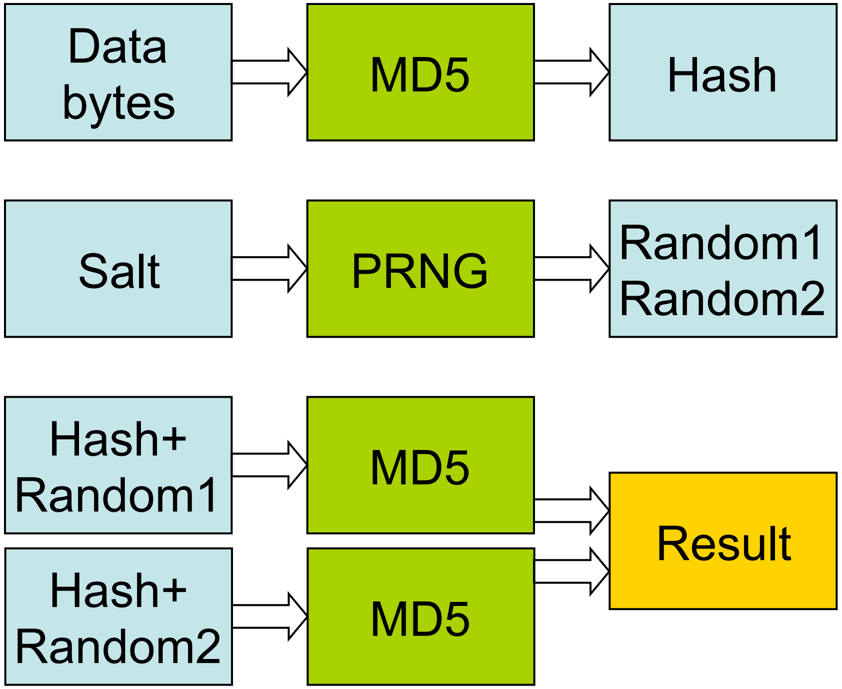
```
typedef struct {
  DWORD id;             // Area ID
  DWORD cbSalt;         // Salt length
  BYTE abSalt[cbSalt];// Salt
  DWORD cbSig;    // Len(Sign(Area))
  BYTE abSig[cbSig];  // Sign(Area)
  DWORD nRange;         // Ranges count
  struct {
    DWORD o;            // Range offset
    DWORD cb;           // Range length
  } r[nRange];          // Array of ranges
} T_ODDv3_Area;
```

ODDv3: Info part structure

```
typedef struct {
  DWORD cbInfo;         // Length of Info part
  DWORD cbSigSalt;      // Image/oddInfo salt length
  BYTE sigSalt[cbSigSalt]; // Image/oddInfo salt
  DWORD v3;             // Version again? == 3
  DWORD cbFile;         // Total size of file
  DWORD vHash;          // 1: VxWorks, 2 or 3: DryOS
  DWORD KeyID;          // Encryption key ID
  DWORD BoardID;        // Board ID
  DWORD KeySalt;        // HMAC key salt
  DWORD nArea;          // Number of areas follows
  T_ODDv3_Area[nArea]; // Area descriptions
  BYTE zeros[];         // Zero filling
} T_ODDv3_Info;
```

ODDv3: Hash algorithm version

Ver 1, OS: VxWorks



Data bytes → MD5 → Hash

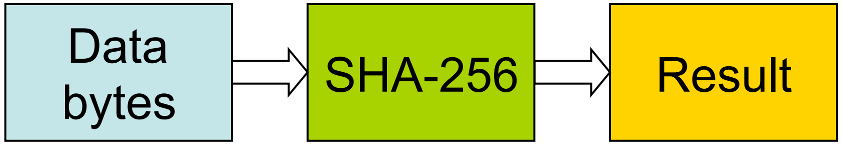
Salt → PRNG → Random1 Random2

Hash+ Random1 → MD5

Hash+ Random2 → MD5

→ Result

Ver 2 and 3, OS: DryOS



Data bytes → SHA-256 → Result

Note: Salt is not used (but still stored in ODD)

ODDv3: Notes on Salt values

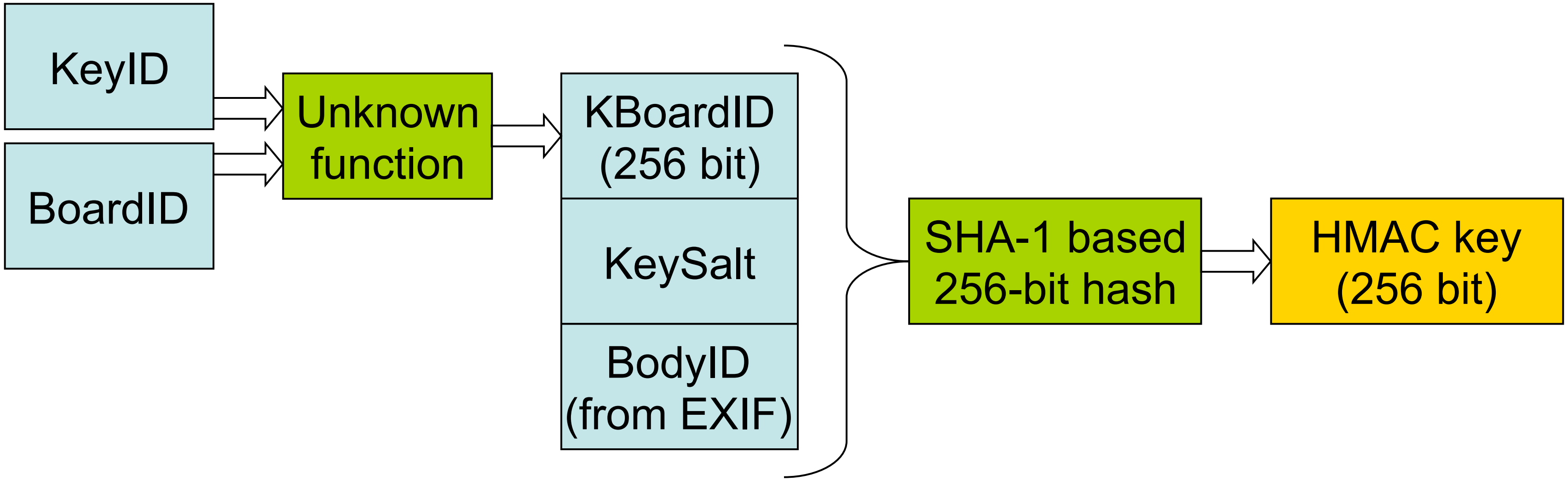
- Salt values are obtained from weak PRNG

```
static DWORD seed;
DWORD randCanon (void) {
  seed = seed * 0x41C64E6D + 0x3039;
  return (seed >> 16) & 0x7FFF;
}
```

- Seed value is based on total number of shots taken by camera (Shutter Counter)
- ODD information could be used to discover actual Shutter Counter value!

ODDv3: HMAC Key

```
DWORD KeyID;       // Encryption key ID
DWORD BoardID;     // Board ID
DWORD KeySalt;     // HMAC key salt
```



ODDv3: Notes on HMAC key

- KBoardID value is depends on KeyID and BoardID values
- KeyID is in range 1..9 (inclusive)
- Every camera uses unique KBoardID

- Knowing KeyID, BoardID and KBoardID triplet allows forging ODD for any camera!
- Key can be extracted from the camera!

Verification devices: DVK-E1

- Introduced with the EOS-1Ds in 2002
- Works in Windows only
- Supports EOS-1Ds only
- Discontinued

Verification devices: DVK-E2

- Introduced in 2004
- Works in Windows only
- Supports: 1Ds, 1Ds Mark II, 1D Mark II, 1D Mark II N, 20D, 30D, 5D
- Discontinued

Verification devices: OSK-E3

- Introduced in 2007
- Works in 32-bit Windows only
- Supports: all ODD-enabled cameras
- Also support images encryption on 1D[s] Mark III+
- Costs about $700

Model name	ODD version	V2 key	Announced
EOS-1D			2001-09-25
EOS-1Ds	probably 1	1	2002-09-24
EOS 10D			2003-02-27
EOS 300D			2003-08-20
EOS-1D Mark II	2	2	2004-01-29
EOS 20D	2	3	2004-08-19
EOS-1Ds Mark II	2	4	2004-09-21
EOS 350D			2005-02-17
EOS 5D	2	5	2005-08-22
EOS-1D Mark II N	2	6	2005-08-22
EOS 30D	2	7	2006-02-21
EOS 400D			2006-08-24

Model name	KeyID seen	vHash	Announced
EOS-1D Mark III		1	2007-02-22
EOS-1Ds Mark III		1	2007-08-20
EOS 40D	1	1	2007-08-20
EOS 450D	2	1	2008-01-24
EOS 1000D	2	1	2008-06-10
EOS 50D	1	2	2008-08-26
EOS 5D Mark II	1	2	2008-09-17
EOS 500D	3	2	2009-03-25
EOS 7D	4	2	2009-09-01
EOS-1D Mark IV		2	2009-10-20
EOS 550D	4	2	2010-02-08
EOS 60D	4	3	2010-08-26

Summary: What we can do?

- Dump camera’s memory
- Run our code on camera’s processor
- Extract secret keys from the camera
- Calculate and verify ODDv2 for models with known key
- Calculate ODDv3 for any camera using known KeyID/BoardID/KBoardID triplet

Summary: What we can’t do [yet]?

- Generate and verify ODDv2 images for models with unknown key
- Calculate KBoardID from KeyID/ BoardID and verify ODDv3 if KBoardID is unknown for given KeyID/BoardID

Summary: What Canon can do?

- With currently available models – nothing
- With future models:
  - Implement HMAC calculation in cryptoprocessor which does not expose secret key
  - Prevent camera from running non-Canon’s code to avoid illegal usage of cryptoprocessor
- Hire people who really understands security :)

Conclusion

- We reported to CERT and Canon on September 21, 2010
- Still no response from Canon…

- Verdict about image originality obtained via Canon’s OSK can’t be relied upon

TROOPERS
The IT-Security Conference

Forging Canon Original Decision Data

Thank you! ;)

Dmitry Sklyarov

ELCOMSOFT
PROACTIVE SOFTWARE