



OBJECTIF SÉCURITÉ
Architecte de la sécurité informatique

Exposing crypto bugs through reverse engineering

Philippe Oechslin





Cracking crypto systems...

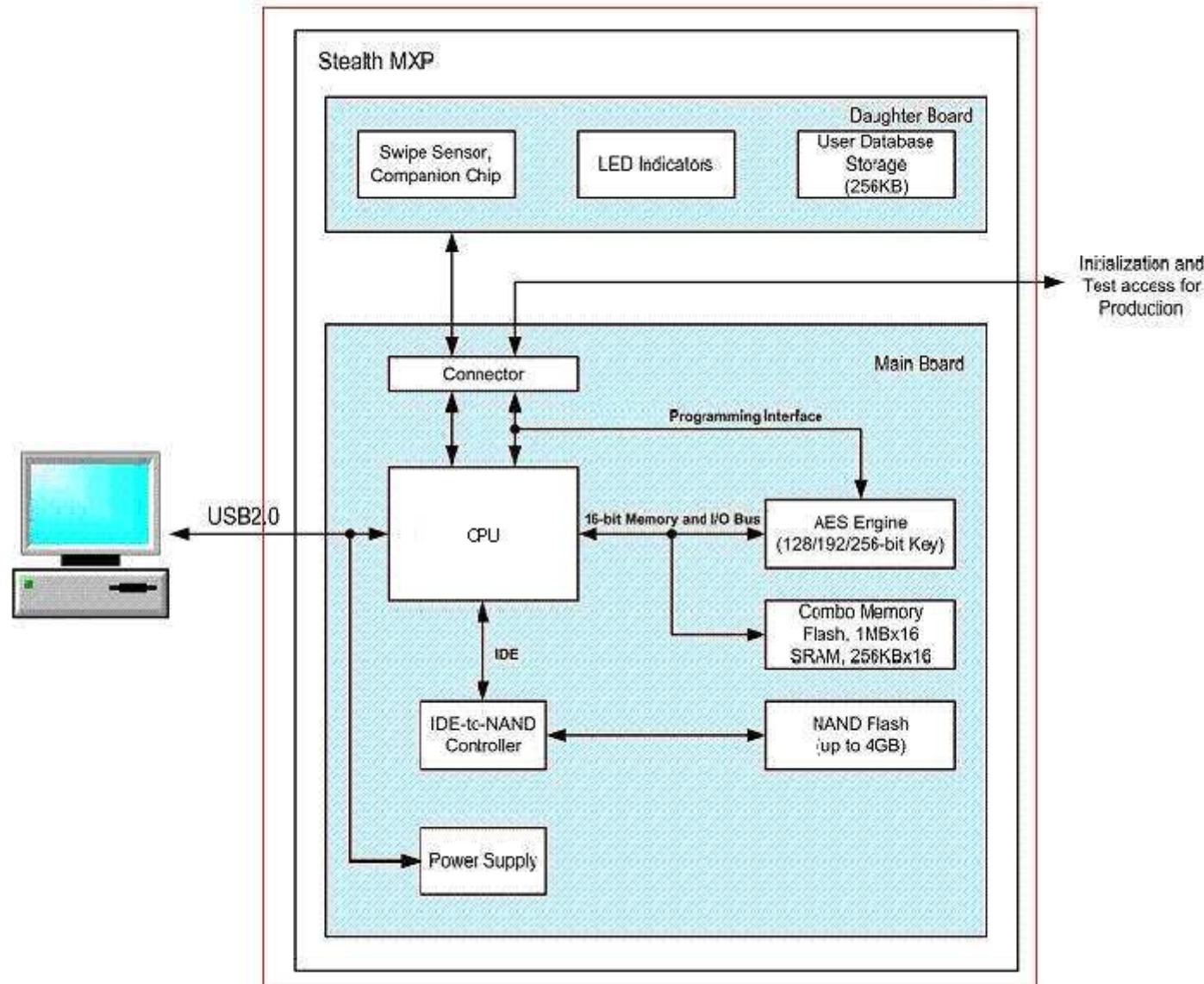
- Cryptanalysis is hard, but so is the correct implementation of cryptography
- A good way to crack a commercial crypto system is thus to reverse engineer it in order to find the implementation errors:
 - MXI Stealth
 - EISST E-capsule PrivateSafe
 - Data Becker Privat Safe



MXI Stealth MXP

- o FIPS 142-3 level 2 certified USB Key
 - AES on-chip encryption
 - Authentication through password (windows application) or fingerprint (OS independent)
 - Upon connection a first removable drive with a locked contents appears
 - Upon successful authentication a second drive appears

MXI Stealth MXP



Password protection

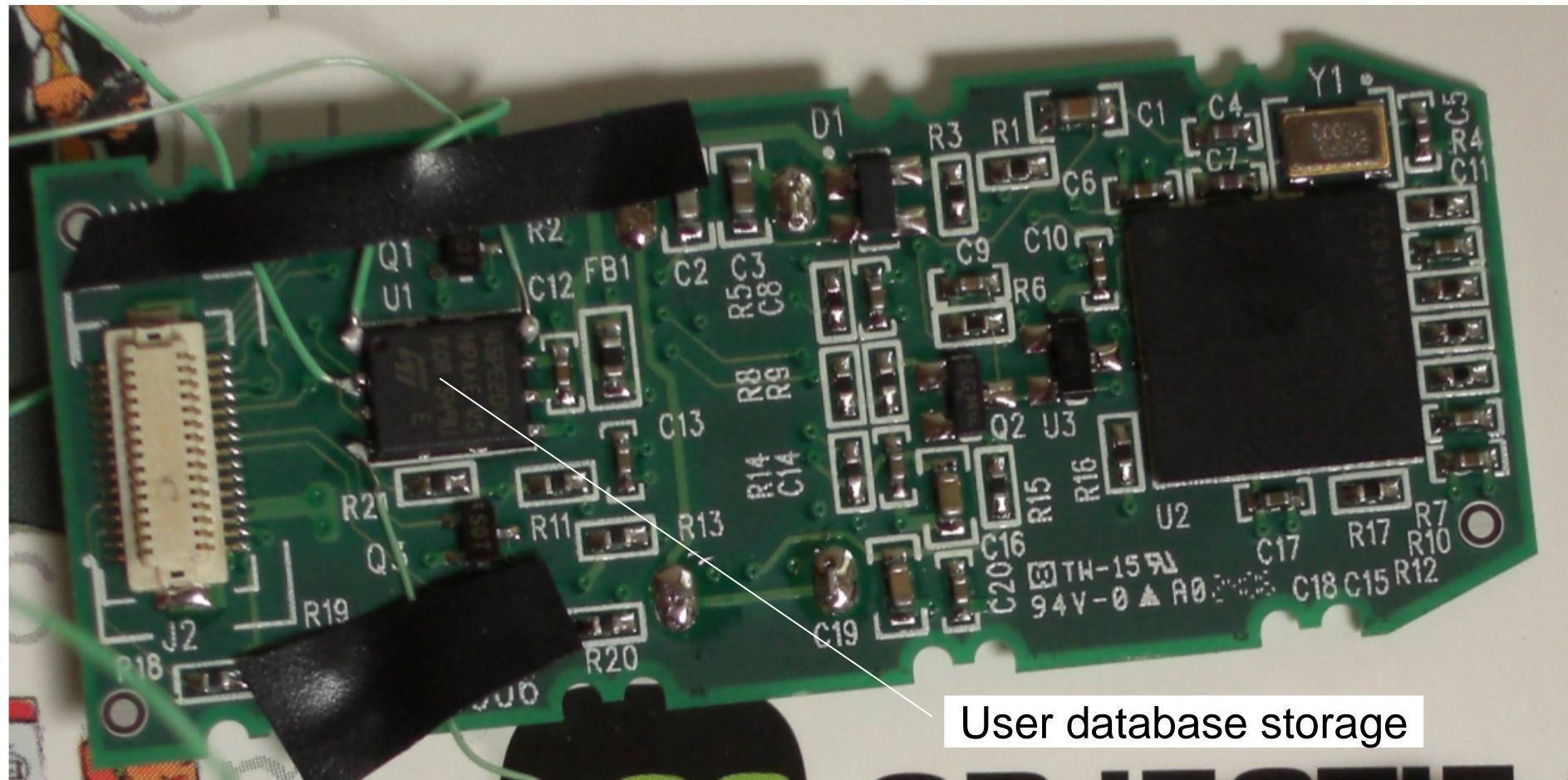


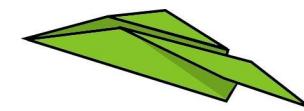
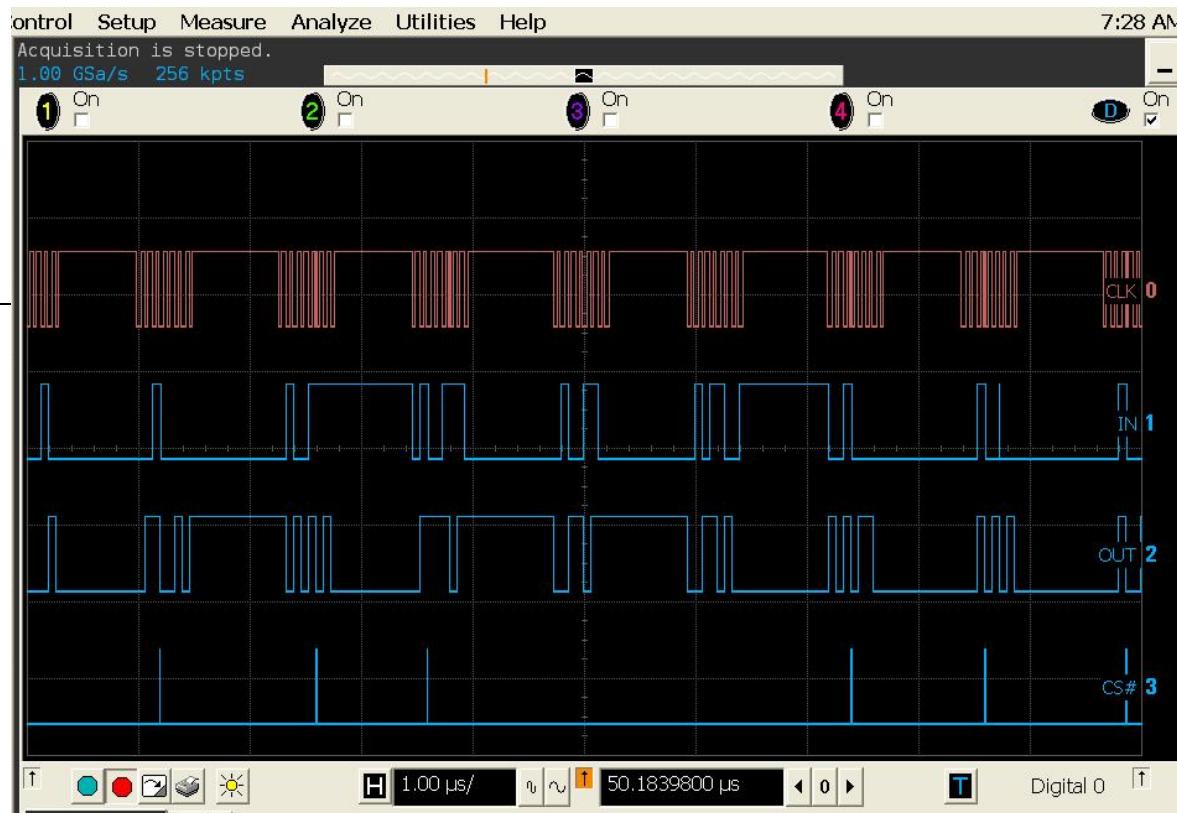
Passwords are injected upon creation from the external USB interface. A random salt is also injected as plain text through the USB interface and stored in EEPROM. The password is combined with the salt, then hashed using the SHA256 algorithm and stored in EEPROM associated with the user. The password is then deleted from memory. Password authentication and verification is done by

Upon a failed password attempt, there is a delay of 500 milliseconds. Note that this delay also applies when a password verification operation is done. This delay allows a maximum of 120 tries per minute. Therefore the probability of a random



Reading the user database





00000000	00 00 00 00 00 00 02 08	20 01 65 aa 7b 37 34 ace.{74.
00000010	2a 26 02 65 aa 7b 37 34	ac 2a 26 76 e1 98 fd 39	*&.e.{74.*&v...9
00000020	3f 31 1b 71 dc b7 78 be	01 d9 32 cc de 14 08 fd	?1.q..x....2.....
00000030	1e 32 7b 68 eb eb 7d 5d	f5 bc 18 20 a8 9b b5 17	.2{h..}].... .
00000040	a8 a1 ad dc be ec 06 da	25 56 ed 01 02 2f 83 18%v..../. .
00000050	2d 7d 5d fe d5 32 12 f4	65 0e 8f b5 00 00 00 00	-}].2..e.....
00000060	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000070	00 00 00 00 00 00 00 00	00 00 00 00 00 03 00 00
00000080	01 fe 00 ff ff ff ff ff	ff ff ff ff ff 5f 41 64Ad
00000090	6d 69 6e 00 00 00 00 00	00 00 00 00 00 00 00 00	min.....
000000a0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000b0	00 00 00 00 00 ff 00 00	00 00 00 00 00 00 00 00
000000c0	08 00 00 00 00 00 00 55	38 37 34 31 32 35 32 35 U87412525
000000d0	35 37 37 32 35 37 39 00	00 00 00 00 00 00 00 00	5772579.....
000000e0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00



Reverse engineering the software

- There is a library to exchange encrypted messages with the key
 - Apparently the password is encrypted and sent to the key
 - After some messages are exchanged the protected disk is activated
- A logging function is implemented.
 - It does not write log messages into a log file but they can bee seen in the memory.
 - A simple patch of the code can reactivate the log file



MXI Demo

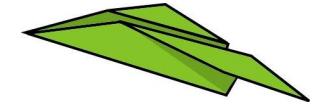
- o What is this blob in the memory???



Surprise

- A blob received from the key has the following content:

D Dump - 03460000..03468FFF															
03465420	FF														
03465430	FF														
03465440	FF														
03465450	FF	FF	FF	FF	53	53	44	5F	42	4C	4F	42	01	00	00
03465460	10	00	00	00	55	34	35	33	38	31	30	37	36	32	30
03465470	38	38	37	35	70	00	00	00	53	53	44	5F	42	4C	4F
03465480	03	00	00	00	08	00	00	00	50	77	64	46	69	72	73
03465490	55	73	65	05	00	00	00	46	41	4C	53	45	09	00	00
034654A0	50	77	64	48	61	73	68	65	73	28	00	00	00	AC	E5
034654B0	90	00	89	51	5C	92	ED	10	00	E9	DF	0E	A6	74	C0
034654C0	A0	ED	40	AE	10	46	5A	43	20	21	8E	A1	82	B1	A5
034654D0	E0	31	8F	83	1C	07	00	00	00	50	77	64	54	69	60
034654E0	04	00	00	00	03	2E	84	48	2C	07	00	00	00	50	77
034654F0	54	69	60	65	04	00	00	00	3B	07	40	48	FF	FF	FF
03465500	FF														
03465510	FF														
03465520	FF														
03465530	FF														
03465540	FF														
03465550	FF														



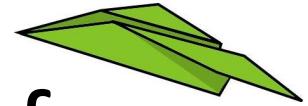
Hashes!

- o The 60 byte string are three SHA1 hashes
 - The current password
 - The previous 2 passwords
- o The "enterprise" version of the software needs this info to make sure the user does not reuse the current and last n passwords
- o This information is received by the software even *before* the user has authenticated



Correcting the issues

- o The validation of the password should have been done in hardware
- o If done in software, the hashes should at least have been stored in a secure area protected by the hardware and only accessible with proper authentication
- o MXI issued a patch within a week to correct the issue



Example: EISST E-capsule PrivateSafe

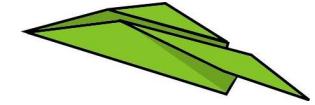


- PrivateSafe is a software that creates encrypted containers
- There are 4 passwords:
 1. The admin password allows managing the container
 2. The public password reveals one part of the content
 3. The private password gives access to the rest of the content
 4. The panic password deletes all files and gives access to an empty container

Files:

- There are 2 files:
 - the encrypted file system
 - a control file

```
philippe@os-philippe$ hd safe.eck
00000000 0000000400000001 30000000000000d0 | .....0.... |
00000010 cc02bff06f1fabd5 0b557c25376e3508 | ....o....U|%7n5. |
00000020 b11896880c63fe4b 7d9267399e3120d9 | .....c.K}.g9.1 . |
00000030 b878da285895a7b5 4e192f7eb22a1454 | .x.(X...N./~.*.T|
00000040 9cb536a459555fdd c38d5a2d724700ec | ..6.YU_...Z-rG.. |
00000050 ae8e827cd88d78b6 f928cf48c948c507 | ...|..x..(.H.H.. |
00000060 35eda128be2f906b dc50b6b43c6a6ccc | 5..(./.k.P..<jl. |
00000070 3cbe31ab1f1c035b 6ce95b6ab7d0f403 | <.1....[1.[j.... |
00000080 f48c08e55cb66501 a051444aed60b679 | ....\e..QDJ.'y |
00000090 68b8324d482e58d6 c8c32e3fe810a75e | h.2MH.X....?...^ |
000000a0 644e3eb29f33427d f65ad86647245014 | dN>..3B}.Z.fG$P. |
000000b0 ac9d77179292688a 38ca3ad869c5ead0 | ...w...h.8.:i... |
000000c0 f47bcce8383d2c26 3896eef1986a2af9 | .{..8=&8....j*. |
000000d0 dc32fbf1702bde8e 3fba5cdb9231b392 | .2..p+..?.\..1.. |
000000e0 0000000131000000 00000060c272e084 | ....1.....'r.. |
000000f0 330ea76a36b22bb0 d5fd646ccad138d8 | 3..j6.+....d1..8. |
00000100 929f55ee16f639a6 a17a94fc539b1b0e | ..U....9..z..S... |
00000110 100d5b03ef2e9992 2a611359a4fc91a0 | ...[.....*a.Y.... |
00000120 1387f4cbc8e9ca7a 832f4887f3d963ab | .....z./H...c. |
00000130 cff256d38cb636c4 7bda1aab9e8911cf | ..V...6.{..... |
00000140 ad3ebcab2539036 2e3a338f00000001 | .>....S.6.:3..... |
00000150 3200000000000060 bd4a85a3e1daffe7 | 2.....'J..... |
00000160 59c91f0c1e1117e1 ea5cafe2901615ef | Y.....\..... |
00000170 7b8c3b8a5d10a3e3 7d9cdd9dbc8b4eea | {.;.}.....N. |
00000180 d26d2735ecb3a04d 6620dd5ae99daae2 | .m'5...Mf .Z.... |
00000190 f2c2848d394bb620 a3afa1bb5a70fafa | ....9K. ....Zp.. |
000001a0 03ae6776738bec3a 3517424f844f1125 | ..gvs...:5.B0.0.% |
000001b0 05e076b76d971562 0000000133000000 | ..v.m..b....3... |
000001c0 0000006048b1012f 47fb2881dd726864 | ...'H../G.(..rhd|
```



Funny crypto

- o Through reverse engineering we find that
 - Each block of the control file is encrypted with AES 256 CTS mode
 - The key is the SHA256 hash of the corresponding password
 - The IV is the ripemd160 hash of the password
 - The clear text of blocks 1,2 and 3 are the same
 - Block 0 corresponds to the admin key



Decryption by hand

```
philippe@os-philippe:~/tmp$ hd block1
```

00000000	7e d2 8b bd c5 26 1e d9	d2 79 c2 e9 5a 5b 68 a7	~....&...y..Z[h.
00000010	3a ba 6c 90 53 ee 24 e7	9c c6 18 2f 6a 81 9c 34	:.1.S.\$..../j..4
00000020	a5 62 b6 6a 8b 1d f4 93	a9 44 20 42 ae 03 1c 0a	.b.j.....D B....
00000030	ff e5 e3 f6 5b dc 06 d7	e2 90 3a 5a 8c c9 fb 68[.....:Z...h
00000040	f3 73 3a 9d 3a 0d 9b 3e	93 b5 a2 4d 32 29 c9 1a	.s:::::>...M2)..
00000050	89 43 99 2c f6 83 aa 0c	f4 71 11 14 05 c1 4e 5f	.C.,.....q....N_
00000060			

```
philippe@os-philippe:~/tmp$ pw=public; cat block1 | openssl enc -aes-256-cbc  
-d -K `echo -n $pw | openssl dgst -sha256 -hex` -iv `echo -n $pw | openssl  
dgst -ripemd160 -hex | cut -b1-32` -nopad | hd
```

00000000	00 00 00 03 00 00 00 08	75 73 65 72 6e 61 6d 65username
00000010	00 00 00 08 50 68 69 6c	69 70 70 65 00 00 00 08Philippe....
00000020	68 61 72 64 77 61 72 65	00 00 00 14 32 8b 0a 83	hardware....2...
00000030	69 f9 de 0d 86 28 23 1b	ab 67 8a d3 22 42 b7 e4	i....(#..g.."B..
00000040	00 00 00 08 73 6f 66 74	77 61 72 65 00 00 00 14software....
00000050	dc ef 7d af 03 85 6f 97	da 4c f9 e0 a8 92 ef 99	..}....o..L.....
00000060			

```
philippe@os-philippe:~/tmp$
```



The four blocks

00 00 00 07 00 00 00 08	75 73 65 72 6e 61 6d 65username
00 00 00 04 75 73 65 72	00 00 00 01 30 00 00 00user....0...
00 00 00 05 61 64 6d 69	6e 00 00 00 00 00 00 01admin.....
31 00 00 00 00 00 00 06	70 75 62 6c 69 63 00 00	1.....public..
00 00 00 01 32 00 00 00	00 00 00 07 70 72 69 762.....priv
61 74 65 00 00 00 00 01	33 00 00 00 00 00 00 05	ate.....3.....
73 68 72 65 64 00 00 00	00 00 00 08 68 61 72 64	shred.....hard
77 61 72 65 00 00 00 14	11 d9 4c 53 80 fa 07 bd	ware.....LS....
ee 88 9e 20 44 e9 ae be	ac 12 41 e5 00 00 00 08	... D.....A.....
8b e6 0b ec 3f 81 04 3e	bf bd 03 6f 76 29 78 ce?..>...ov)x.
a8 2b be 0d 64 98 08 15	1b 3e 2c 91 00 6b d7 b0	.+..d....>,...k..
00 00 00 03 00 00 00 08	75 73 65 72 6e 61 6d 65username
00 00 00 04 75 73 65 72	00 00 00 08 68 61 72 64user....hard
77 61 72 65 00 00 00 14	11 d9 4c 53 80 fa 07 bd	ware.....LS....
ee 88 9e 20 44 e9 ae be	ac 12 41 e5 00 00 00 08	... D.....A.....
53 08 5b fc 7c ff 75 d9	00 db ac a7 ad 26 8d 07	S.[.. .u.....&..
ae f8 42 df 69 9c 3e 8b	83 44 27 04 3f c7 ea 7e	..B.i.>..D'.?..~
00 00 00 03 00 00 00 08	75 73 65 72 6e 61 6d 65username
00 00 00 04 75 73 65 72	00 00 00 08 68 61 72 64user....hard
77 61 72 65 00 00 00 14	11 d9 4c 53 80 fa 07 bd	ware.....LS....
ee 88 9e 20 44 e9 ae be	ac 12 41 e5 00 00 00 08	... D.....A.....
90 0c 92 84 09 51 6b 85	04 e4 1c 17 30 2a 96 a0Qk.....0*..
86 40 1d ec 3d be 2c 01	2b 7b a2 27 8d 14 b0 02	.@..=.,.+{.'....
00 00 00 03 00 00 00 08	75 73 65 72 6e 61 6d 65username
00 00 00 04 75 73 65 72	00 00 00 08 68 61 72 64user....hard
77 61 72 65 00 00 00 14	11 d9 4c 53 80 fa 07 bd	ware.....LS....
ee 88 9e 20 44 e9 ae be	ac 12 41 e5 00 00 00 08	... D.....A.....
fb 83 31 52 36 8a a4 c7	95 a9 34 68 4b 7a aa f7	..1R6.....4hKz..
b2 8e 16 6a 68 26 b4 84	bc 61 81 5c ee 63 6f a8	...jh&....a.\.co.

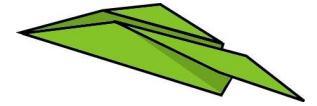


The trick

- Exchanging two blocks in the control file inverts the role of their keys
 - e.g. private \leftrightarrow public
- Worse
 - Shred \leftrightarrow private!
- Actually, exchanging just the single ascii characters that identify the blocks is enough..
- But... the software works as advertized

```
philippe@os-philippe$ hd safe.eck
000000d0 dc32fbf1702bde8e 3fba5cdb9231b392
000000e0 0000000131000000 00000060c272e084
000000f0 330ea76a3692bb0 d5fd646ccad138d8
00000100 929f55ee16f639a6 a17a94fc539b1b0e
00000110 100d5b03ef2e9992 2a611359a4fc91a0
00000120 1387f4cbc8e9ca7a 832f4887f3d963ab
00000130 cff256d38cb636c4 7bda1aab9e8911cf
00000140 ad3ebcab2539036 2e3a338f00000001
00000150 32000000000000060 bd4a85a3e1daffe7
00000160 59c91f0c1e1117e1 ea5cafe2901615ef
00000170 7b8c3b8a5d10a3e3 7d9cdd9dbc8b4eea
00000180 d26d2735ecb3a04d 6620dd5ae99daae2
00000190 12c2848d394bb620 a3afa1bb5a70fafa
000001a0 03ae6776738bec3a 3517424f844f1125
000001b0 05e076b76d971562 0000000133000000
```





EISST demo:

- Accessing the private data with the shred password

```
*I:\Documents\PrivateSafe\safe.eck - Notepad++
Fichier Edition Recherche Affichage Format Langage Paramétrage Macro Exécution Plugins Document ?
safe.eck

1 NUL NUL NUL EOT NUL NUL SOHO NUL NUL NUL NUL NUL NUL °t« ETX™ FSá FS¥ DC4 áC» þBŠ & °ÈéHi1) ± BSE
2 óD÷_ "P3^ áSO! ózDNF@ DLEh" ÈæU±, Mœ6¹ Ów< Á[ NUL NUL NUL SOH1 NUL NUL NUL NUL NUL NUL d~Óç kíë RSÙ Òy
3 ýåöö [ ÜACK x á: ZCEÙhós : ñ:
4 >> "µcM2) ÉSUB% C™, öf² FFôq DC1 DC4 ENQÁN_n5éä NUL NUL NUL SOH3 NUL NUL NUL NUL NUL NUL dW» ESCç Tí
5 4Tü! Y' ÜENQOBSSiy,, ÚIH) àk" + E- i ADC3 NUL NUL NUL SOH2 NUL NUL NUL NUL NUL NUL NUL dÈx- kíy~ ñk*t@
6 ÇY- «ýiM² à DLE² žó ýô ¥> šú ñOfç ! O² tê BS SOH Üx HšÜ< W# " 4€üä _ CX0 IÜ CEADC1 ; ü° " É' DC3 a«+ ØSçR> +AQ-ZÉ
```

DataBecker PrivateSafe



Gute software muss nicht teuer sein...

- This software creates an encrypted data container
- The control block is at the beginning of the file
- The encrypted file system is encrypted with blowfish CBC in blocks of 4096 bytes
- The IV of all blocks is the same
- The key is the user password
- The first 8 bytes of a block are its number
 - They are not ordered but the numbers are small (lots of 0 bits)





DataBecker demo:

- o What does this routine do?

C CPU - main thread, module safe

Address	OpCode	Mnemonic	Operands
004040C9	\$ 55	PUSH EBP	
004040CA	. 8BEC	MOV EBP,ESP	
004040CC	. 51	PUSH ECX	
004040CD	. C745 FC 000000	MOV DWORD PTR SS:[EBP-4],0	
004040D4	.^EB 09	JMP SHORT safe.004040DF	
004040D6	> 8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]	
004040D9	. 83E8 01	SUB EAX,1	
004040DC	. 8945 08	MOV DWORD PTR SS:[EBP+8],EAX	
004040DF	> 837D 08 00	CMP DWORD PTR SS:[EBP+8],0	
004040E3	.^74 22	JE SHORT safe.00404107	
004040E5	. 8B4D FC	MOV ECX,DWORD PTR SS:[EBP-4]	
004040E8	. D1E1	SHL ECX,1	
004040EA	. 894D FC	MOV DWORD PTR SS:[EBP-4],ECX	
004040ED	. 8B55 0C	MOV EDX,DWORD PTR SS:[EBP+C]	
004040F0	. 33C0	XOR EAX,EAX	
004040F2	. 8A02	MOV AL,BYTE PTR DS:[EDX]	
004040F4	. 8B4D FC	MOV ECX,DWORD PTR SS:[EBP-4]	
004040F7	. 33C8	XOR ECX,EAX	
004040F9	. 894D FC	MOV DWORD PTR SS:[EBP-4],ECX	
004040FC	. 8B55 0C	MOV EDX,DWORD PTR SS:[EBP+C]	
004040FF	. 83C2 01	ADD EDX,1	
00404102	. 8955 0C	MOV DWORD PTR SS:[EBP+C],EDX	
00404105	.^EB CF	JMP SHORT safe.004040D6	
00404107	> 8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]	
0040410A	. 8BE5	MOV ESP,EBP	
0040410C	. 5D	POP EBP	
0040410D	. C3	RETN	

Registers (FPU)

Register	Value	Description
EAX	00252BA0	
ECX	00000008	
EDX	00252BA4	ASCII "0sec2009"
EBX	00000000	
ESP	0012F970	
EBP	0012F984	
ESI	00437730	safe.00437730
EDI	0012FB04	
EIP	004040C9 <safe.checksum>	
C	0 ES 0023 32bit 0(FFFFFFFF)	
P	0 CS 001B 32bit 0(FFFFFFFF)	
A	0 SS 0023 32bit 0(FFFFFFFF)	
Z	0 DS 0023 32bit 0(FFFFFFFF)	
S	0 FS 003B 32bit 7FFDF000(FFF)	
T	0 GS 0000 NULL	
D	0 0 LastErr ERROR_SUCCESS (00000000)	
EFL	00000202 (NO,NB,NE,A,NS,PO,GE,G)	
ST0	empty +UNORM 5848 0012EDE8 F4CDA908	
ST1	empty +UNORM 0001 02010E3D BF8157F5	
ST2	empty 3.9990710863206502160e-4055	
ST3	empty +UNORM 5848 0012ED78 F4CDA934	
ST4	empty 5.7746097852815439430e-3835	
ST5	empty 1.0000000000000000000000000000000	



Checksum !

- Before decrypting the file system the software verifies that the key matches a checksum...
- The checksum is made by xor-ing all characters of the password shifted by one bit each.

Password	Ascii	
t	116	0 1 1 1 0 1 0 0
e	101	0 1 1 0 0 1 0 1
s	115	0 1 1 1 0 0 1 1
t	116	0 1 1 1 0 1 0 0
1	49	0 0 1 1 0 0 0 1
2	50	0 0 1 1 0 0 1 0
3	51	0 0 1 1 0 0 1 1
4	52	0 0 1 1 0 1 0 0
checksum	2B72	0 0 1 0 1 0 1 1 0 1 1 0 1 1 0 0 1 0



Attack

- The checksum gives us:
 - A minimum of the pw length
 - 1 bit per character of the pw
 - The last character of the pw
- Simple attack:
 - Generate all passwords, verify the checksum before attempting decryption
- Efficient attack
 - Generate only passwords that match the checksum
 - Use two alphabets (even and odd parity)
 - Use the alphabet corresponding the previous characters and the checksum
 - Calculate the last character from the checksum

```

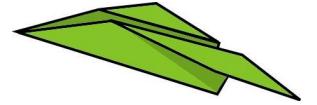
for (i2=0; i2<HALF_SETSIZE; i2++) {
    key[2]=i2[((ck&0x40)?chars1:chars0)];
    ck^=(key[2]<<6);

    for (i1=0; i1<HALF_SETSIZE; i1++) {
        key[1]=i1[((ck&0x80)?chars1:chars0)];
        ck^=(key[1]<<7);

        key[0] = ck>>8;

        if (key[0] > 96) { /* only lowercase letter as first */
            Blowfish_Init (&ctx, (unsigned char*)key, 9);
            L = L0, R=R0;
            Blowfish_Decrypt(&ctx, &L, &R);
            if (((R&0xffffffff)==0x00543210) &&
                ((L&0xffffffff)==0x00dcba98)) {
                printf("\n%d: Key found!!!!!! --> %s <---",n,key);
                printf(" plaintext: %08x %08x\n\n",
                       L^0xfedcba98, R^0x76543210);
                exit(9);
            }
        }
    }
}

```



Result

- Blowfish has a complex key setup
 - Key setup is very slow
 - 25'000 pws per second on a single core
- For a 9 character alphanum password we gain a factor of 15'872 in time
- In our case we cracked a nine lowercase letter password in 2.5 hours instead of 1.7 years



Quiz ...

- o Besides their funny cryptographic specialities, all shown systems have a fundamental crypto flaw
 - Which flaw is it?
- o None of them uses a random salt or initialisation vector!
- o We could thus build Rainbow Tables for each of this systems



Conclusions

- o Crypto is hard to implement correctly
 - Who would have thought this...
- o If no source code is given, only reverse engineering can find the errors
- o When possible, prefer open source solutions over closed source, they are easier (and cheaper) to verify.