Reverse engineering of CHIASMUS from GSTOOL

It hurts.

Jan Schejbal

Outline

- Intro (with some history)
- Issues *ow, ow, ow, ow, ow, ow, ow, ow, …*
- Disclosure/Timeline
- CHIASMUS cipher

INTRO

Credits

- We made this:
  Erik Tews, Julian Wälde, and me

- independent discovery by Felix Schuster
- ECB first discovered by Christian Gresser

About me

- studied IT Security in Darmstadt
- spend too much time behind a keyboard
- found SSL issue (and more) in AusweisApp
  - BSI probably hates me anyways ;-)
- I know **applied** cryptography
  - implementations, not ciphers themselves

The BSI

- Bundesamt für Sicherheit in der Informationstechnik
- Federal Office for Information Security
- emerged 1991 from BND **crypto department**
  - *Zentralstelle für das Chiffrierwesen (ZfCh)*
  - *1989: Zentralstelle für Sicherheit in der Informationstechnik (ZSI)*

BSI ciphers

- non-public (per policy)
- CHIASMUS algorithm for software
  - Chiasmus for Windows/Linux (restricted)
  - GSTOOL (freely accessible)
- LIBELLE algorithm for hardware
  - PLUTO chip → SINA boxes
    - Development started ca. 1996/97
      http://www.zeit.de/1998/09/krypt.txt.19980219.xml

CHIASMUS algorithm

- Not completely unreasonable
  - First reference to CfW 1.0: October 2000
  - Rijndael published 1998
  - AES announced November 2001
  - GPG v1.0 released 1999
  - US crypto export restrictions lifted in January 2000
  - Wikipedia was founded 2001

Chiasmus for Windows

- File encryptor
  - likely created by BSI
  - uses CHIASMUS algorithm
  - format obfuscated
  - CBC (claimed)
  - asks user for randomness

➔ Probably not affected!

GSTOOL

- Management of IT baseline protection process
- Made by contractors
- Encryption function to protect export databases
  - uses CHIASMUS algorithm
  - can encrypt any file

GSTOOL versions

- CHIASMUS encryption: since v3.0
  - unchanged since then (until 4.8 in 2013)
  - released ca. July 2003 [1)]
  - developed by contractor (Steria-Mummert)
- Future version: 5.0
  - massive contractor fail (PERSICON) [2)]
  - was planned but now cancelled

1) https://www.datenschutzzentrum.de/systemdatenschutz/archiv_meldung/sm16.htm
2) http://www.heise.de/ix/meldung/BSI-verweigert-Abnahme-des-GSTOOL-5-0-1824176.html

ISSUES

Implementing crypto

You are given a secure algorithm (e.g. AES).

What can you mess up?

Implementing crypto

You are given a secure algorithm (e.g. AES).
What can you mess up?

- Key generation
- Block cipher mode
- Integrity checking

Key generation

- Key must be
  - random
  - unpredictable
  - long enough
- use good randomness (/dev/random)
- use a good PRNG if you must (/dev/urandom)

Key generation (the GSTOOL way)

```
uint8_t key[104]; // pointless length
srand(time(NULL));
for (int i = 0; i < 104; i++) {
        key[i] = rand() & 0xff;
}
```

- Spot the problem.

Key generation (the GSTOOL way)

```
uint8_t key[104]; // pointless length
srand(time(NULL));
for (int i = 0; i < 104; i++) {
        key[i] = rand() & 0xff;
}
```

- Non-cryptographic PRNG
- 32 bits of state
- 32 bits of seed
- predictable seed

Consequences

- Key space smaller than 2^32
  - Brute-force attack possible
  - less than 1 hour (unoptimized)
  - optimization possible (Felix says 24 bits)
- Key generation time = key
  - reduces attacks to a few minutes
- **Encryption worthless & dangerous**
  - false sense of security

Key generation (the proper way)

- CryptGenRandom (Windows API, since Win9x)
- /dev/(u)random

- „Can we use this tutorial?“ http://eternallyconfuzzled.com/arts/jsw_art_rand.aspx
  - How to properly convert a time_t to an integer for calling srand() in a platform-independent way
  - Mentions that rand() gives bad randomness

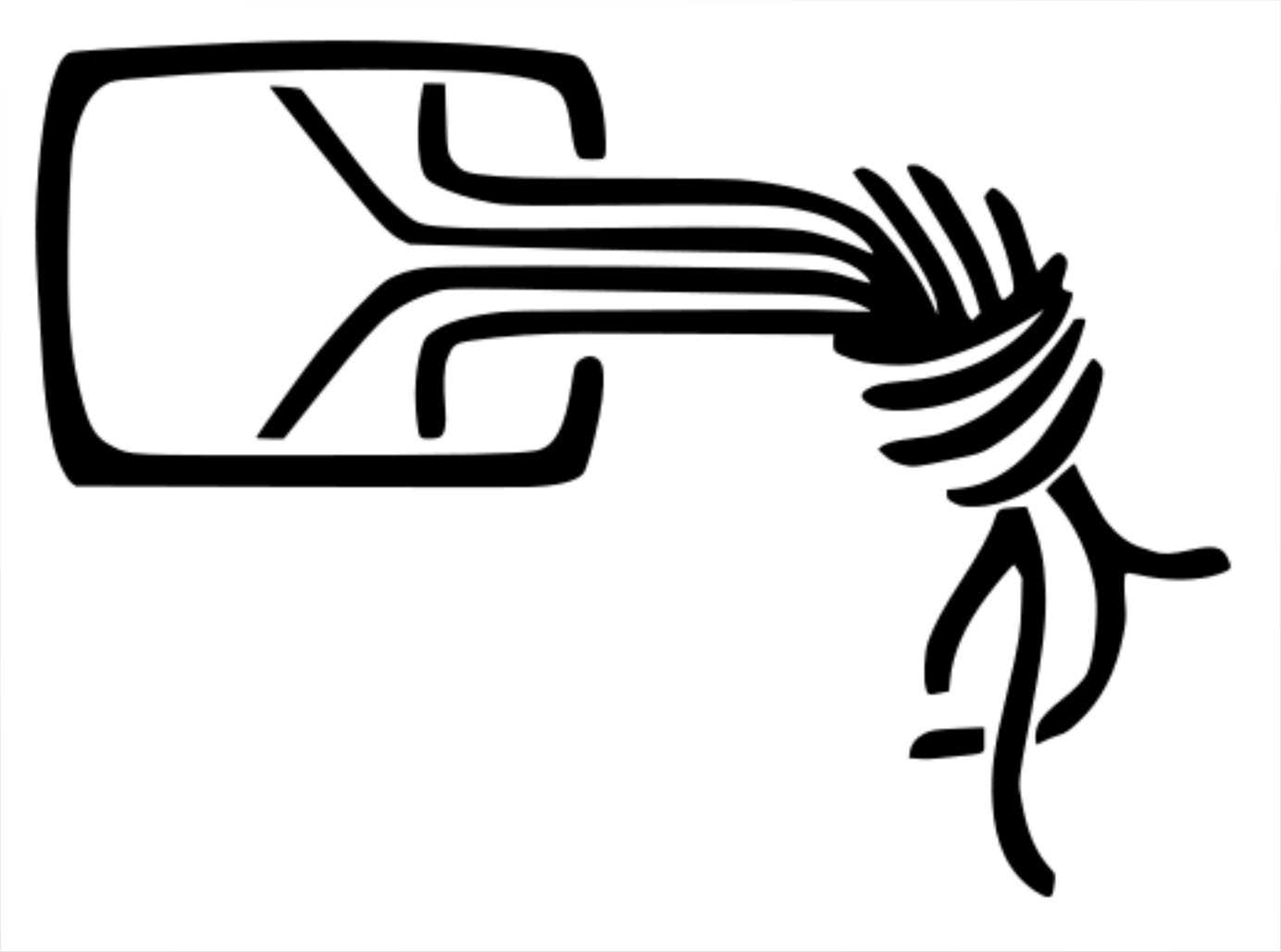
Block cipher mode

- ECB encrypts every block separately
- Same plaintext = same ciphertext
- Leaks information
  - Attacker can see identical parts in file
  - Attacker can see two files are identical
- Theoretical?

Block cipher mode - CBC

Block cipher mode - ECB

Block cipher mode - Plain

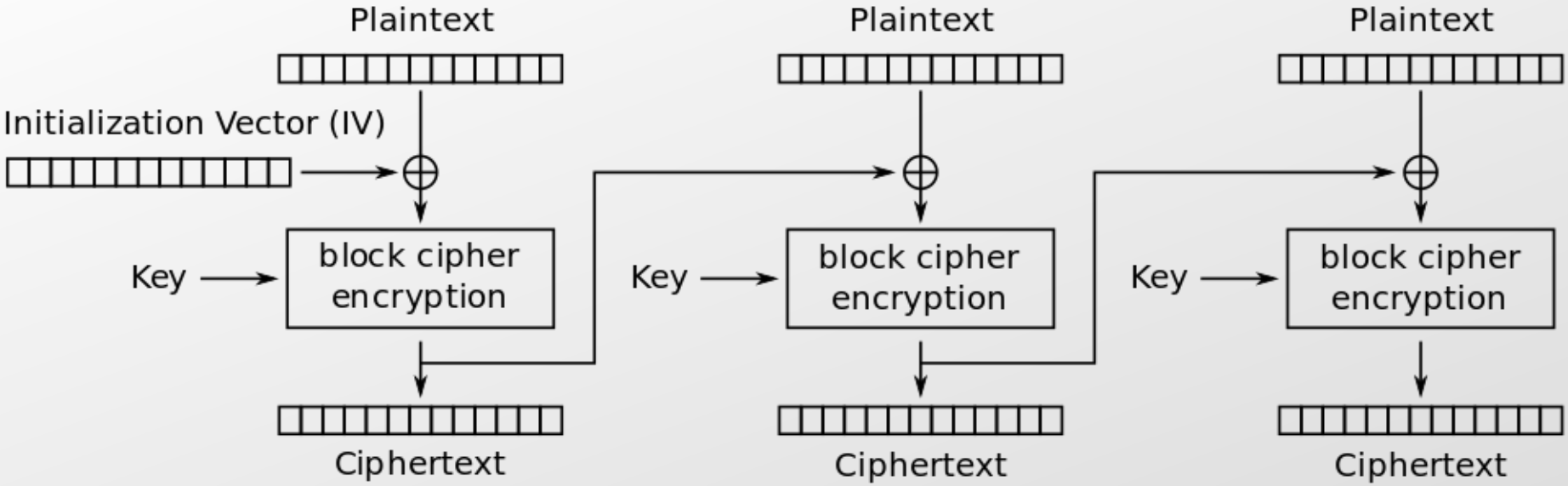


Block cipher mode

- Entire block must be identical to get same result
- Not too vulnerable:
  - Photos (with noise)
  - Compressed data
- Vulnerable:
  - Raw bitmaps with large solid areas
  - Databases

Solution: e.g. CBC

- Use random value (IV) to start
- Use last output (random!) to randomize next



Cipher Block Chaining (CBC) mode encryption

Source: Wikipedia, http://commons.wikimedia.org/wiki/File:CBC_encryption.svg, public domain

Integrity checking

- Encryption does not guarantee integrity
- People will rely on it
- ECB: reorder blocks, truncate, …
- CBC: truncate (front and back!)
- stream ciphers/counter mode: bit flipping(!)
  - exploited in the wild (WEP)

Integrity: Practical example

- Attacker wants to make a file look authentic
- RAR has „header“ at the end
  - ignores garbage before archive
- Make victim encrypt evilrar || decoy
- obtain ciphertext
- truncate to get correctly encrypted evilrar
  - must match block boundaries

Integrity checking

- Hash in file not sufficient
- Truncation attacks

  evil || hash(evil) || decoy || real hash
- Stream ciphers: easy if plaintext known
  - calculate original hash
  - calculate new hash
  - flip bits that need flipping

0	1	0	1	0	0	1	0	1	1	0	1	1	0	1
0	0	0	1	1	0	1	0	1	1	1	1	0	1	1

Integrity checking

- Do it properly
  - use high-level library if possible
- (H)MAC with separately-derived key
- Authenticated modes (e.g. GCM)

Key file sanity checks

- No check if key matches file
  - yields garbage if wrong file is used
  - usability nightmare, not security critical
- insecure if wrong file is used for encryption

Summary

- Encryption in GSTOOL broken
  - in nearly every possible way
  - *except* the cipher itself
- Practical key recovery (full break) with minimal effort
- Chiasmus software tool (CfW/CfL) likely **not** affected

DISCLOSURE

Timeline

- **2008-09-24** ECB discovered by Christian Gresser
  http://www.mitternachtshacking.de/blog/727-snake-oil-alarm-chiasmus-im-gstool
- **2009** Analysis by Felix Schuster, contacted BSI beforehand, got legal threats, did not report
- **2011-09-19** hack.lu CTF – GSTOOL-Chiasmus used as challenge and solved (possibly reported)
- **2011** Our analysis

Timeline

- **2011-11-14** Contacted the BSI, demonstrated issue
  - later: exact issue description (via telephone)
- **2011-11-25** Vendor advisory, announced service pack with fix “soon”
  - advisory is hard to find today, only anonymous credit
- Further communication, advice on fixing the issues, provided BSI with draft of paper
- **2011-12-06** received legal threats

Legal threats

„Vielen Dank für die Arbeit, die Sie in die weitere Analyse gesteckt haben. Ich muss Sie jedoch darauf aufmerksam machen, dass Reverse engineering auch von BSI-Produkten einen Urheberrechtsverstoß darstellt und von unseren Juristen sehr strikt verfolgt wird. Ich bitte daher dringend darum, dass das Dokument in unserem kleinen Kreis bleibt. Können Sie mir das bitte bestätigen?“

“Thank you for the work you have put into further analysis. However, I must point out that reverse engineering BSI products constitutes a breach of **copyright** and will be prosecuted very strictly by our lawyers. I therefore urgently request that the document remains confidential.
Can you please confirm that?”

- later defused in a phone call

Legal issues

- § 69c UrhG protects software
- § 69d(3) allows “observing” the program
- § 69e allows decompiling for interoperability purposes (with limitations)
- Algorithms usually not copyrightable (in DE), c.f. e.g. § 69a(2)
- Code was not reused, no decompiling

→ **lots of wasted time and no definitive answer**

Timeline

- delayed publication, further communication, several requests for an update on the status
- **2013-06-06** Service pack released (“silently“)
- **2013-07-22** UbiCrypt Summer School in Bochum with lecture by Felix Schuster
  http://prezi.com/bzyvzzdsxtkm/ubicrypt-chm/
- **2013-09-11** Publication of our advisory
- **2013-12-02** FOIA request
  https://fragdenstaat.de/anfrage/ifg-anfrage-gstool/

“Fix”

- Disable encryption button
  - probably the best choice
  - bad UI implementation
  - insufficient warnings to users
- Hoping for GSTOOL5
  - not going to happen

Should we keep it secret?

- Security *and* obscurity can be OK
  - not having the algorithm does make analysis hard
  - the “bad guys” (NSA & Co.) know about this
    - link CHIASMUS—GSTOOL easy to find
    - Reverse Engineering is easy
- BSI had lots of time to fix and notify
  - very slow progress
  - over 2 years since original advisory

Impact

- Users insufficiently notified
  - advisory hard to find today
  - patch notes not very clear
- GSTOOL used for generic file encryption
- Embarrassing:
  - Awareness of issues was lower in 2003, but:
  - CBC invented 1976

Lessons learned

- Set **hard** deadlines (< 1 year…) for publication
  - especially if issues easy to fix
  - may speed up fix development
- Personal policy:
  Legal threats = immediate publication
  - aim of legal threats is often to avoid publication
    → publication removes motivation for threat
  - YMMV, you may get sued
  - full disclosure may save you a lot of hassle

CHIASMUS

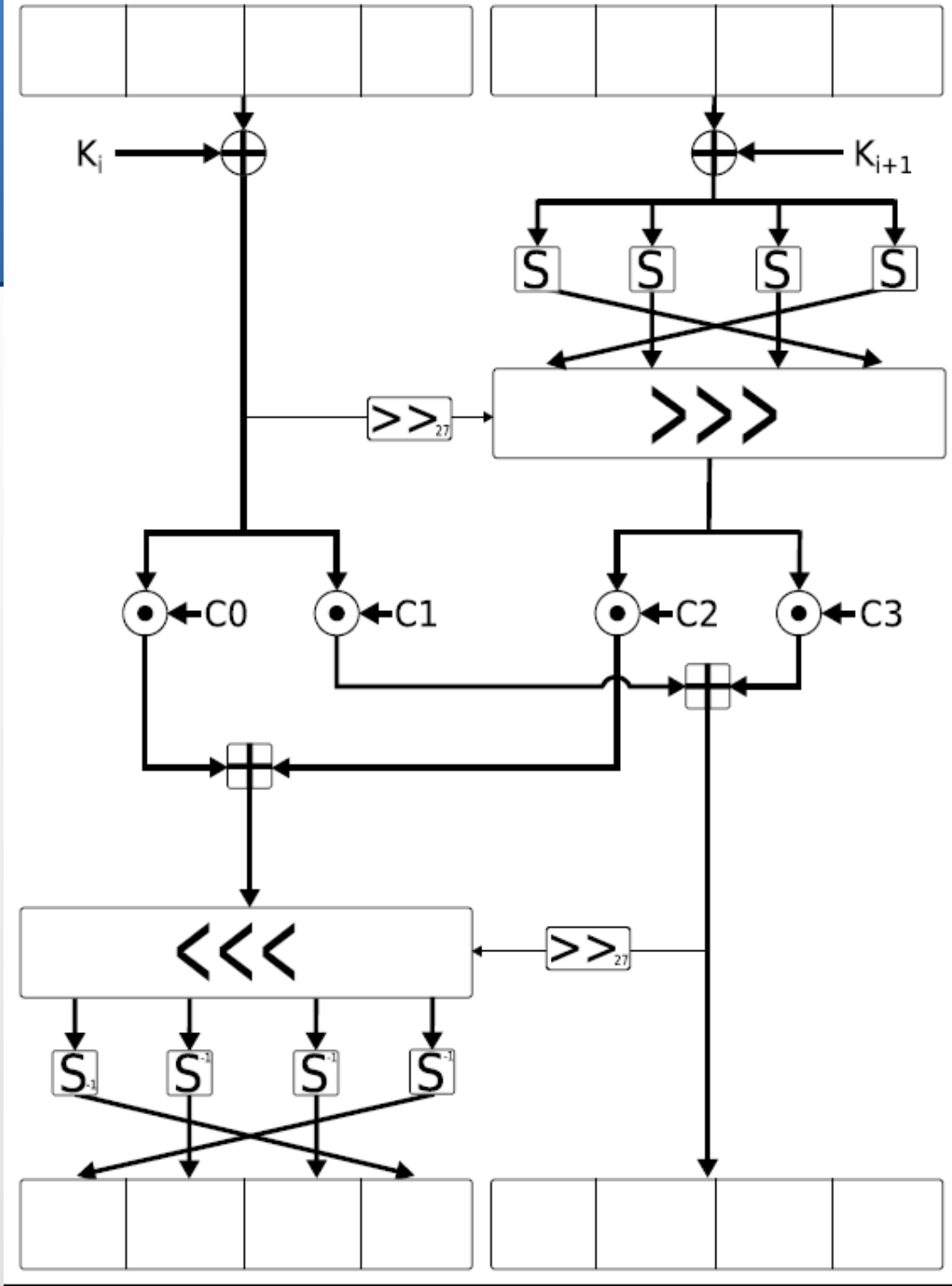
CHIASMUS itself

- Block cipher
- Key size 160 bits
  - CfW: allegedly 128 + 32bit checksum
  - expanded to 13*64 bit in key schedule
    - equals 104 bytes generated (not used) in GSTOOL
- Block size 64 bits
- Appears optimized for 32-bit CPUs
- 12 rounds + 1 key addition

CHIASMUS round

- Key addition (XOR)
- 8-bit S-Box
- byte shuffling
- variable shift
- matrix multiplication
- variable shift
- S-Box (inverse)
- byte shuffling

for encryption:
c0 = 0xb06553a6, c2 = 0xfad512b5,
c1 = 0xb846cb9d, c3 = 0x4614965b



S-Boxes are inverse

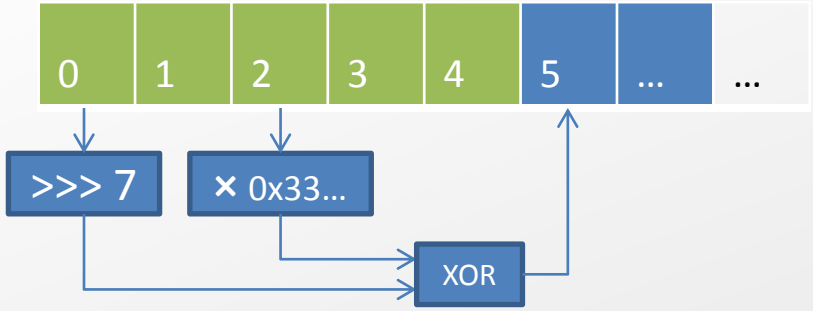
```
uint8_t SBOX1[256] = {
    0x65, 0x33, 0xCF, 0xB9, 0x37, 0x64, 0xCD, 0xF3, 0x26, 0x3A, 0xC1, 0xA2, 0x72, 0x8A, 0x8F, 0xE3,
    0xFD, 0x56, 0xB3, 0x0F, 0x10, 0x2B, 0x3E, 0xA0, 0xBD, 0x1E, 0xAB, 0x1D, 0x9C, 0xE2, 0x87, 0x98,
    0xA8, 0xD3, 0xB4, 0xDF, 0x92, 0x75, 0x3B, 0x39, 0x20, 0xA5, 0xFA, 0x1B, 0xBE, 0x90, 0xF6, 0x09,
    0xE5, 0x61, 0xC4, 0xC9, 0x06, 0xC2, 0xA6, 0x1C, 0xF9, 0x94, 0x7B, 0x53, 0x73, 0x01, 0x25, 0x9A,
    0x1A, 0xFF, 0xE9, 0x5A, 0x76, 0x13, 0x4B, 0x95, 0xAC, 0x0B, 0xC7, 0xB2, 0xB8, 0xD6, 0x17, 0xA9,
    0x27, 0xEB, 0xD1, 0x5C, 0xC3, 0x9B, 0x22, 0x15, 0x8E, 0x40, 0x11, 0x5E, 0x57, 0x16, 0xD0, 0xB0,
    0x5D, 0x79, 0x31, 0xBB, 0xEA, 0x4F, 0xD9, 0xDE, 0x00, 0x0A, 0xD7, 0xAD, 0x3F, 0x99, 0x68, 0x34,
    0x66, 0xF0, 0x44, 0x35, 0x89, 0x54, 0x81, 0xB1, 0x84, 0x2A, 0x8B, 0x6F, 0xC0, 0x43, 0xFE, 0x96,
    0x48, 0x82, 0x0C, 0xDA, 0x74, 0xBC, 0x21, 0xF1, 0x67, 0x2E, 0xDB, 0x49, 0xE4, 0xD5, 0x71, 0x59,
    0x29, 0xE0, 0xA1, 0x30, 0xDD, 0x91, 0x6B, 0xB7, 0xB6, 0x69, 0xC5, 0x80, 0xAA, 0x6D, 0xA3, 0x2C,
    0x05, 0x78, 0xBA, 0x51, 0x14, 0x07, 0xD4, 0xEC, 0x7E, 0xCC, 0x24, 0x62, 0x9E, 0xDC, 0x8C, 0xD8,
    0x1F, 0x46, 0xE8, 0x9F, 0x4E, 0xA4, 0x85, 0x32, 0xCE, 0xA7, 0xFC, 0xE1, 0x97, 0xAE, 0x2D, 0x52,
    0x7D, 0x0E, 0x6C, 0x83, 0x5F, 0xBF, 0x18, 0x7C, 0x36, 0x63, 0x0D, 0xEF, 0xC8, 0x5B, 0x55, 0x12,
    0x4A, 0xF2, 0x70, 0x38, 0xF8, 0xAF, 0x86, 0x77, 0x47, 0x04, 0x23, 0x02, 0x6E, 0x4C, 0x58, 0x03,
    0x50, 0x7A, 0x3D, 0x28, 0xF5, 0xE7, 0x41, 0xF4, 0x45, 0x60, 0x6A, 0x08, 0x88, 0x7F, 0x9D, 0x93,
    0x4D, 0xD2, 0x2F, 0xEE, 0xE6, 0xCB, 0xED, 0xFB, 0xCA, 0xF7, 0x19, 0xB5, 0x42, 0x8D, 0xC6, 0x3C,
};
```

S-Boxes are inverse

```
uint8_t SBOX2[256] = {
    0x68, 0x3D, 0xDB, 0xDF, 0xD9, 0xA0, 0x34, 0xA5, 0xEB, 0x2F, 0x69, 0x49, 0x82, 0xCA, 0xC1, 0x13,
    0x14, 0x5A, 0xCF, 0x45, 0xA4, 0x57, 0x5D, 0x4E, 0xC6, 0xFA, 0x40, 0x2B, 0x37, 0x1B, 0x19, 0xB0,
    0x28, 0x86, 0x56, 0xDA, 0xAA, 0x3E, 0x08, 0x50, 0xE3, 0x90, 0x79, 0x15, 0x9F, 0xBE, 0x89, 0xF2,
    0x93, 0x62, 0xB7, 0x01, 0x6F, 0x73, 0xC8, 0x04, 0xD3, 0x27, 0x09, 0x26, 0xFF, 0xE2, 0x16, 0x6C,
    0x59, 0xE6, 0xFC, 0x7D, 0x72, 0xE8, 0xB1, 0xD8, 0x80, 0x8B, 0xD0, 0x46, 0xDD, 0xF0, 0xB4, 0x65,
    0xE0, 0xA3, 0xBF, 0x3B, 0x75, 0xCE, 0x11, 0x5C, 0xDE, 0x8F, 0x43, 0xCD, 0x53, 0x60, 0x5B, 0xC4,
    0xE9, 0x31, 0xAB, 0xC9, 0x05, 0x00, 0x70, 0x88, 0x6E, 0x99, 0xEA, 0x96, 0xC2, 0x9D, 0xDC, 0x7B,
    0xD2, 0x8E, 0x0C, 0x3C, 0x84, 0x25, 0x44, 0xD7, 0xA1, 0x61, 0xE1, 0x3A, 0xC7, 0xC0, 0xA8, 0xED,
    0x9B, 0x76, 0x81, 0xC3, 0x78, 0xB6, 0xD6, 0x1E, 0xEC, 0x74, 0x0D, 0x7A, 0xAE, 0xFD, 0x58, 0x0E,
    0x2D, 0x95, 0x24, 0xEF, 0x39, 0x47, 0x7F, 0xBC, 0x1F, 0x6D, 0x3F, 0x55, 0x1C, 0xEE, 0xAC, 0xB3,
    0x17, 0x92, 0x0B, 0x9E, 0xB5, 0x29, 0x36, 0xB9, 0x20, 0x4F, 0x9C, 0x1A, 0x48, 0x6B, 0xBD, 0xD5,
    0x5F, 0x77, 0x4B, 0x12, 0x22, 0xFB, 0x98, 0x97, 0x4C, 0x03, 0xA2, 0x63, 0x85, 0x18, 0x2C, 0xC5,
    0x7C, 0x0A, 0x35, 0x54, 0x32, 0x9A, 0xFE, 0x4A, 0xCC, 0x33, 0xF8, 0xF5, 0xA9, 0x06, 0xB8, 0x02,
    0x5E, 0x52, 0xF1, 0x21, 0xA6, 0x8D, 0x4D, 0x6A, 0xAF, 0x66, 0x83, 0x8A, 0xAD, 0x94, 0x67, 0x23,
    0x91, 0xBB, 0x1D, 0x0F, 0x8C, 0x30, 0xF4, 0xE5, 0xB2, 0x42, 0x64, 0x51, 0xA7, 0xF6, 0xF3, 0xCB,
    0x71, 0x87, 0xD1, 0x07, 0xE7, 0xE4, 0x2E, 0xF9, 0xD4, 0x38, 0x2A, 0xF7, 0xBA, 0x10, 0x7E, 0x41,
};
```

Key schedule

- Operates on 32-bit integers
- Original key = first 5 integers



- Add position to key integers (except first 5)

```
#define RROT(x,amount) (((x) >>
    (amount)) | ((x) << (32-(amount))))
void keyschedule(uint8_t* keybytes) {
   uint32_t* keyints =
    (uint32_t*)keybytes;
   for (int i = 0; i < 21; i++) {
      uint32_t val = keyints[i];
      val = RROT(val,7);
      uint32_t val2 = keyints[i+2];
      val = val^(val2*0x33333333);
      keyints[i+5] = val;
   }
   for (int i = 5; i< 26; i++) {
      keyints[i] += i;
   }
}
```

Security

- Cipher looks reasonably secure
  - not the usual snake-oil
- **Purely theoretical** attack:
  - related-key attack on 5-round reduced round version with complexity 2^{140}

Is this CHIASMUS?

- ***Probably.***
- Cipher likely provided by BSI
  - contractor lacks crypto knowledge
- Separate cipher unlikely
- could differ from “regular” CHIASMUS
  - possibility explicitly mentioned with LIBELLE
  - Constants, S-Boxes, number of rounds, … (possibly even key schedule)
- Programming mistakes possible

SUMMARY

Summary

- CHIASMUS looks OK
  - now public, analyze it!
- Implementation in GSTOOL **fully broken**
  - BSI most likely did not review this
  - Government IT contractors suck
- Do not implement crypto yourself
  - not even using AES

Questions to YOU

- Do you know how GSTOOL users were notified?
- Any examples of users relying on it?

Thank you
for your attention

Q&A