How to break Petya's cipher with pen and paper

Francisco Blas Izquierdo Riera AKA klondike

About me

- Security interested since 17
- Computer Engineer & MSc
- Gentoo Hardened developer
- Cryptography fan:
  - Implemented AES-SIV in an Atmega (Arduino) bootloader
  - Implemented CTR, CMAC and SIV modes in the Haskell crypto-api library
  - Wrote own efficient TTH implementations
  - Pushed for adding stronger cryptography to the ADC protocol
- Currently working as pentester and providing cryptographic support at Coresec Systems AB

Short intro to crypto

- Confusion: ability of a cipher to hide the relation between plain and ciphertext
- Diffusion: ability of a cipher to apply a bit change to all its outputs
- Stream cipher: a cipher able to encrypt data bit by bit
- Salsa20: a stream cipher, it encrypts a known state which is xored with the plaintext
- ChaCha20: a derivative of Salsa20. Used nowadays as a replacement for RC4 in TLS and SSH

The backstory

- Petya published, first ransomware working from the boot sector
- Leostone discovers a flaw on the keying system
    - Key entropy reduced to 46.03 bits (from 92.06)
- Leostone attempts a brute force attack
    - Discarded for being too slow
- Leostone implements a genetic algorithm search
    - It works

The Petya's cipher challenge

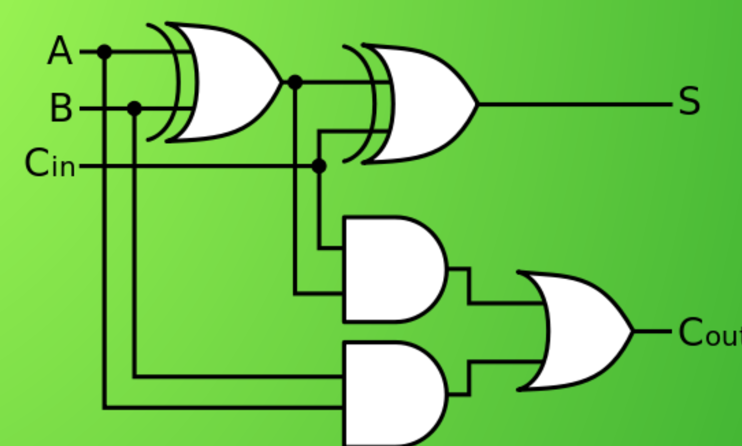
- Petya uses a Salsa20 like cipher
- Petya's cipher was broken using genetic algorithms
  - Unknown relation between plaintext and ciphertexts (bad confusion)
- Was this an issue specific to Petya or did it affect Salsa20?
- If it affects Salsa20, does it also affect ChaCha20?

The cryptanalysis constraints

- Fully independent research
  - Carried out in spare time
  - Used own tools
  - Only pen and paper available most of the time

The algebraic approach

- Results on procedure to break cipher for all keys
- Models cipher as set of equations
  - Adds → groups of xors, ands and ors
  - Rotates → remap bits
  - Xors → xor of each bit



Equations get complicated soon

A more abstract approach

- Focus on diffusion and confusion
- How do inputs contribute to outputs?
- Heuristic of words with less contributions

4,0,12 5,1,11 6,2,12 7,3,13

8, 0,4 ~~[illegible]~~ 0,12

12, 0,4, 0,12, 4,0,12

Example

Word	Contributors	Word	Contributors
0	0	8	8
1	1	9	9
2	2	10	10
3	3	11	11
4	4	12	12
5	5	13	13
6	6	14	14
7	7	15	15

Before the first run

Example

Word	Contributors	Word	Contributors
0	0	8	8
1	1	9	9
2	2	10	10
3	3	11	11
4	0,4,12	12	12
5	5	13	13
6	6	14	14
7	7	15	15

After:

```
u := uint32(me[0] + me[12])
me[4] ^= uint16(u<<7 | u>>(32-7))
```

Example

Word	Contributors	Word	Contributors
0	0,4,8,12	8	0,4,8,12
1	1	9	9
2	2	10	10
3	3	11	11
4	0,4,12	12	0,4,8,12
5	5	13	13
6	6	14	14
7	7	15	15

After:

```
u = uint32(me[4] + me[0])
me[8] ^= uint16(u<<9 | u>>(32-9))
u = uint32(me[8] + me[4])
me[12] ^= uint16(u<<13 | u>>(32-13))
u = uint32(me[12] + me[8])
me[0] ^= uint16(u<<18 | u>>(32-18))
```

Example

Word	Contributors	Word	Contributors
0	0,4,8,12	8	0,4,8,12
1	1,5,9,13	9	1,5,9
2	2,6,10,14	10	2,6,10,14
3	3,11,15	11	3,7,11,15
4	0,4,12	12	0,4,8,12
5	1,5,9,13	13	1,5,9,13
6	2,6,10,14	14	6,10,14
7	3,7,11,15	15	3,7,11,15

After the first row round

Example

Word	Contributors	Word	Contributors
0	0,1,2,3,4,5,6,8,9,10,11,12,13,14,15	8	0,4,8,12
1	0,1,3,4,5,8,9,11,12,13,15	9	1,5,9
2	0,1,2,3,4,5,6,8,9,10,11,12,13,14,15	10	2,6,10,14
3	0,1,2,3,4,5,6,8,9,10,11,12,13,14,15	11	3,7,11,15
4	0,4,12	12	0,4,8,12
5	1,5,9,13	13	1,5,9,13
6	2,6,10,14	14	6,10,14
7	3,7,11,15	15	3,7,11,15

After the first column quarter round

Example

Word	Contributors	Word	Contributors
0	0,1,2,3,4,5,6,8,9,10,11,12,13,14,15	8	0,1,2,3,4,5,6,7,8,9,10,11,12,14,15
1	0,1,3,4,5,8,9,11,12,13,15	9	0,1,2,3,4,5,6,7,8,9,10,11,12,14,15
2	0,1,2,3,4,5,6,8,9,10,11,12,13,14,15	10	0,1,2,3,4,5,6,7,8,9,10,11,12,14,15
3	0,1,2,3,4,5,6,8,9,10,11,12,13,14,15	11	1,2,3,5,6,7,9,10,11,14,15
4	0,1,2,3,4,5,6,7,9,10,11,12,13,14,15	12	0,3,4,6,7,8,10,11,12,14,15
5	0,1,2,3,4,5,6,7,9,10,11,12,13,14,15	13	0,1,3,4,5,6,7,8,9,10,11,12,13,14,15
6	0,1,2,4,5,6,9,10,12,13,14	14	0,1,3,4,5,6,7,8,9,10,11,12,13,14,15
7	0,1,2,3,4,5,6,7,9,10,11,12,13,14,15	15	0,1,3,4,5,6,7,8,9,10,11,12,13,14,15

After the first column round

Example

Word	Contributors	Word	Contributors
0	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	8	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
1	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	9	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
2	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	10	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
3	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	11	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
4	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	12	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
5	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	13	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
6	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	14	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
7	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	15	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

After the second row round

WOW! That also escalated fast

Back to the design board

- Lots of type casting going on

```
u := uint32(me[0] + me[12])
me[4] ^= uint16(rotl(u,7))
```

- What is the type of the addition in Go?
- Are sign bits expanded on 32-bit cast?
- Why is rotation only 32-bit operation?

Turns out the king had been naked all along!

- Additions returned the same type of their inputs
  - 16-bit unsigned integers
- The 32-bit cast results in zero expansion
- No sign bits to care about
- 0 xor x = x
- Rotations result in unmodified bits

$O_{15} \cdot 1$ $x_0 + x_1$

$O_9 \; x_0 + x_1 \cdot O_7$

9 bits + 7 O's

7 bits + 9 O's

~~1~~ 3 bits + 13 O's

14 O's + 2 bits

5 bits

The first attempt

- Left and right rotations
  - Smallest left: 7 bits
  - Only right: 2 bits
- 5 bits per word unmodified
- Entropy reduced to 176 bits (from 256)

Not Good Enough!

Let’s focus on unmodified bits!

- Given a word:
  - Mark any bit affected by other word as tainted
  - Find non tainted bits

Example

Word	Unmodified bits	Word	Unmodified bits
0	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	8	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
1	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	9	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
2	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	10	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
3	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	11	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
4	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	12	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
5	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	13	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
6	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	14	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
7	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	15	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

Before the run

Example

Word	Unmodified bits	Word	Unmodified bits
0	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	8	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
1	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	9	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
2	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	10	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
3	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	11	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
4	0,1,2,3,4,5,6	12	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
5	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	13	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
6	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	14	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
7	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	15	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

After:

```
u := uint32(me[0] + me[12])
me[4] ^= uint16(u<<7 | u>>(32-7))
```

Example

Word	Unmodified bits	Word	Unmodified bits
0	2,3,4,5,6,7,8,9,10,11,12,13,14,15	8	0,1,2,3,4,5,6,7,8
1	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	9	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
2	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	10	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
3	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	11	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
4	0,1,2,3,4,5,6	12	0,1,2,3,4,5,6,7,8,9,10,11,12
5	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	13	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
6	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	14	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
7	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	15	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

After:
```
u = uint32(me[4] + me[0])
me[8] ^= uint16(u<<9 | u>>(32-9))
u = uint32(me[8] + me[4])
me[12] ^= uint16(u<<13 | u>>(32-13))
u = uint32(me[12] + me[8])
me[0] ^= uint16(u<<18 | u>>(32-18))
```

Example

Word	Unmodified bits	Word	Unmodified bits
0	2,3,4,5,6,7,8,9,10,11,12,13,14,15	8	0,1,2,3,4,5,6,7,8
1	0,1,2,3,4,5,6,7,8,9,10,11,12	9	0,1,2,3,4,5,6
2	0,1,2,3,4,5,6,7,8	10	2,3,4,5,6,7,8,9,10,11,12,13,14,15
3	0,1,2,3,4,5,6	11	0,1,2,3,4,5,6,7,8,9,10,11,12
4	0,1,2,3,4,5,6	12	0,1,2,3,4,5,6,7,8,9,10,11,12
5	2,3,4,5,6,7,8,9,10,11,12,13,14,15	13	0,1,2,3,4,5,6,7,8
6	0,1,2,3,4,5,6,7,8,9,10,11,12	14	0,1,2,3,4,5,6
7	0,1,2,3,4,5,6,7,8	15	2,3,4,5,6,7,8,9,10,11,12,13,14,15

After the first row round

Example

Word	Unmodified bits	Word	Unmodified bits
0	2,3,4,5,6,7,8,9,10,11,12,13,14,15	8	0,1,2,3,4,5,6,7,8
1	0,1,2,3,4,5,6	9	0,1,2,3,4,5,6
2	0,1,2,3,4,5,6,7,8	10	2,3,4,5,6,7,8,9,10,11,12,13,14,15
3	0,1,2,3,4,5,6	11	0,1,2,3,4,5,6
4	0,1,2,3,4,5,6	12	0,1,2,3,4,5,6
5	2,3,4,5,6,7,8,9,10,11,12,13,14,15	13	0,1,2,3,4,5,6,7,8
6	0,1,2,3,4,5,6	14	0,1,2,3,4,5,6
7	0,1,2,3,4,5,6,7,8	15	2,3,4,5,6,7,8,9,10,11,12,13,14,15

After the first column round (and after each successive iteration)

At least 7 bits per word

State entropy reduced to 108 bits

How are the state elements mapped?

This comment is particularly revealing

Const0 uint16 me0!
Key0   uint16 me1
Key2   uint16 me2
Key4   uint16 me3
Key6   uint16 me4
Const2 uint16 me5!
Nounce0 uint16 me6!
Nounce2 uint16 me7!
Counter uint32 me8! me9!
Const4 uint16 me10!
Key8   uint16 me11
Key10   uint16 me12
Key12   uint16 me13
Key14   uint16 me14
Const6 uint16 me15!

Example

Word	Unmodified bits	Word	Unmodified bits
Const0	2,3,4,5,6,7,8,9,10,11,12,13,14,15	Counter_LSB	0,1,2,3,4,5,6,7,8
Key0	0,1,2,3,4,5,6	Counter_MSB	0,1,2,3,4,5,6
Key2	0,1,2,3,4,5,6,7,8	Const4	2,3,4,5,6,7,8,9,10,11,12,13,14,15
Key4	0,1,2,3,4,5,6	Key8	0,1,2,3,4,5,6
Key6	0,1,2,3,4,5,6	Key10	0,1,2,3,4,5,6
Const2	2,3,4,5,6,7,8,9,10,11,12,13,14,15	Key12	0,1,2,3,4,5,6,7,8
Nonce0	0,1,2,3,4,5,6	Key14	0,1,2,3,4,5,6
Nonce2	0,1,2,3,4,5,6,7,8	Const6	2,3,4,5,6,7,8,9,10,11,12,13,14,15

After mapping elements to words

Key entropy is 68 bits

Some questions left

- How is state combined with plaintext?
  - Xor
- How is passphrase expanded?
  - uint16(v<<1)<<8 | uint16(v+'z')
- Which are valid inputs?
  - 123456789abcdefghijkmnopqrstuvwxABCDEFGHJKLMNPQRSTUVWX
- How are the expansions mapped to keys?
  - Letter 0 to Key0, letter 2 to Key2, letter 4 to Key4...

Example

Word	Unmodified bits	Word	Unmodified bits
Const0	2,3,4,5,6,7,8,9,10,11,12,13,14,15	Counter_LSB	0,1,2,3,4,5,6,7,8
e(Letter0)	0,1,2,3,4,5,6	Counter_MSB	0,1,2,3,4,5,6
e(Letter2)	0,1,2,3,4,5,6,7,8	Const4	2,3,4,5,6,7,8,9,10,11,12,13,14,15
e(Letter4)	0,1,2,3,4,5,6	e(Letter8)	0,1,2,3,4,5,6
e(Letter6)	0,1,2,3,4,5,6	e(Letter10)	0,1,2,3,4,5,6
Const2	2,3,4,5,6,7,8,9,10,11,12,13,14,15	e(Letter12)	0,1,2,3,4,5,6,7,8
Nonce0	0,1,2,3,4,5,6	e(Letter14)	0,1,2,3,4,5,6
Nonce2	0,1,2,3,4,5,6,7,8	Const6	2,3,4,5,6,7,8,9,10,11,12,13,14,15

After mapping the passphrase to words

Combining the attacks

- We have 7 bits (or even 9) of the key unmodified
- Plaintext is a string of 0x37s
- Can we infer 'v' from the last 8 bits of: uint16(v<<1)<<8 | uint16(v+'z') ?
  - v = (((w&0xff)⊕0x37)-0x7a)&0xff
- Will it work with the character set and 7 bits?

Yes as no 'v' has the MSB set

Cryptanalysis conclusions:

- We derive this equation:
  - $v=(((w\&0x7f)\oplus 0x37)-0x7a)\&0x7f$
- We use the equation to map words of ciphertext to input key parts
- We dismiss unused key part

We can break Petya with a single plain text

We can even bruteforce lost nonces!

Is Salsa20 broken?

Why?

- Salsa20 has no passphrase mapping at all
  - No passphrase mapping flaw
- Salsa20 uses 32-bit words everywhere
  - No rotation flaw

What happened afterwards?

- Developers released a new version using full Salsa20
  - With subtly broken passphrase mapping
  - Hasherezade published the reverse engineered code and procrash broke it using GPUs
- Released yet another version with more complicated passphrase mapping
  - So far it hasn’t been cracked

Things to take home

- Salsa20 is still safe
- Specific implementations may not be
  - Avoid implementing your own cryptography!
- Cryptanalysis is not always about advanced mathematics.
- The contributors measurement can be used to find in $O(n*m)$ with memory use of $O(n^2)$
  - Too tedious to carry out on pen and paper bit by bit
- The unmodified bits measurement can be used in $O(m)$ with memory use of $O(n)$
  - Works even with pen and paper (cross bits in a matrix as they get modified)

Thanks!

- To Leostone for the implementation, previous work and cryptanalysis challenge
- To my mother and father for supporting my curiosity since I was a kid
- To the SEC-T organizers for making this talk and conference possible
- To those who supported me during the research
- To those who provided input during the preparation of this talk:
  - Huzaifa Essajee
  - Meredith Patterson
  - Hasherezade
  - Tero Hänninen
  - Niklas Andersson
  - Mikael Johansson
- Coresec Systems AB for providing me a place to rehearse this talk
- But especially, to you for your attention