

# Introduction to Cryptographic Attacks

Henning

<henning+ccchh@e-gehirn.de>

January 21, 2015

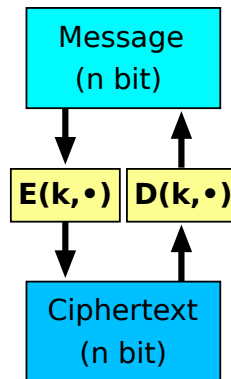
- 1 **Crypto Basics**
  - Cryptographic Primitives
  - XOR and the One Time Pad
- 2 **Attacks**
  - Two Time Pad
  - Birthday Attack
  - CBC Padding Attack
  - Length-extension on Merkle-Damgård
  - Timing attacks
  - Weaknesses in MSCHAPv2
- 3 **HITCON Crypto Challenges**
  - RSA with related messages
  - Combined Collision on short MD5/SHA1
- 4 **Conclusion and Outlook**

# Outline

- 1 **Crypto Basics**
  - Cryptographic Primitives
  - XOR and the One Time Pad
- 2 Attacks
  - Two Time Pad
  - Birthday Attack
  - CBC Padding Attack
  - Length-extension on Merkle-Damgård
  - Timing attacks
  - Weaknesses in MSCHAPv2
- 3 HITCON Crypto Challenges
  - RSA with related messages
  - Combined Collision on short MD5/SHA1
- 4 Conclusion and Outlook

# Block Ciphers

- A block cipher takes a block (fixed numbers of bits) and applies a permutation.
- The permutation is selected by the key.
- Formally: Two algorithms ( $E, D$ ) for encryption/decryption:  
 $E, D: K \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ .  
 $D$  is the inverse of  $E$ .
- Chosen permutation is not efficiently distinguishable from a random permutation.



# Cryptographic Hash functions

A hash function maps a large bit string to a fixed size bit string.

## Properties:

**Pre-image resistance:** Given  $h$ , difficult to find any message  $m$  such that  $h = H(m)$ .

**Second pre-image resistance:** Given  $m_1$ , difficult to find another  $m_2$  such that  $m_1 \neq m_2$  and  $H(m_1) = H(m_2)$

**Collision resistance:** Difficult to find two different messages  $m_1$  and  $m_2$  such that  $H(m_1) = H(m_2)$

Most hash functions fulfill even stronger assumptions, e.g. they behave like a random oracle.

# Outline

## 1 Crypto Basics

- Cryptographic Primitives
- XOR and the One Time Pad

## 2 Attacks

- Two Time Pad
- Birthday Attack
- CBC Padding Attack
- Length-extension on Merkle-Damgård
- Timing attacks
- Weaknesses in MSCHAPv2

### 3 HITCON Crypto Challenges

- RSA with related messages
- Combined Collision on short MD5/SHA1

## 4 Conclusion and Outlook

# eXclusive OR (XOR)

XOR is addition modulo 2:

$a$	$b$	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

## Properties:

- ① Associative:  $(a \oplus b) \oplus c = a \oplus (b \oplus c)$
- ② Commutative:  $a \oplus b = b \oplus a$
- ③ Neutral element: 0. Thus  $a \oplus 0 = 0 \oplus a = a$
- ④ Self-inverse:  $a \oplus a^{-1} = a \oplus a = 0$

# One Time Pad

**Main idea:** XORing an arbitrarily distributed bit string with a uniformly distributed bit string results in a uniformly distributed bit string.

How the One Time Pad works:

- 1 Choose a random key as long as the message.
- 2 XOR the message with the key.

⇒ The resulting ciphertext can't be distinguished from a random bit string!

## Warning

Never use a key twice! It's called **One** Time Pad for good reason.



# Outline

- 1 Crypto Basics
  - Cryptographic Primitives
  - XOR and the One Time Pad
- 2 Attacks
  - Two Time Pad
  - Birthday Attack
  - CBC Padding Attack
  - Length-extension on Merkle-Damgård
  - Timing attacks
  - Weaknesses in MSCHAPv2
- 3 HITCON Crypto Challenges
  - RSA with related messages
  - Combined Collision on short MD5/SHA1
- 4 Conclusion and Outlook

# Two Time Pad

Assume we have two ciphertexts encrypted with the same pad:

$$c_0 = m_0 \oplus k$$

$$c_1 = m_1 \oplus k$$

What information can we derive?

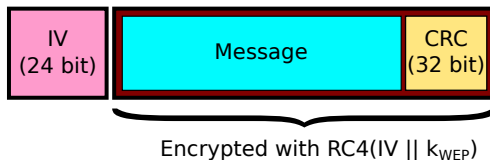
$$c_0 \oplus c_1 = (m_0 \oplus k) \oplus (m_1 \oplus k) = m_0 \oplus m_1 \oplus k \oplus k = m_0 \oplus m_1$$

# XORed English Plaintext

Take a look at the following ASCII characters:

Character	Bit string
<i>SPACE</i>	0 1 0 0 0 0 0
A	1 0 0 0 0 0 1
a	1 1 0 0 0 0 1
B	1 0 0 0 0 1 0
b	1 1 0 0 0 1 0
...	...

Assume  $m_0 = \text{"AbBa"}$  and  $m_1 = \text{"____"} (4 \text{ spaces})$ , then  
 $m_0 \oplus m_1 = \text{"aBbA"}$ .



- RC4 works like the One Time Pad with a generated pad
- Given the same key  $k$ , the same pad is generated
- Repeating IV means same key  $k$ .
- There are  $2^{24} \approx 16.7$  million IVs.

It was not specified how to generate an IV. Many network cards use a counter, some of them reset it to zero after power cycle. That way you get a collision for an IV quickly.

Note: The real world attacks on WEP were related key attacks on RC4, not the two time pad.

# Outline

- 1 Crypto Basics
  - Cryptographic Primitives
  - XOR and the One Time Pad
- 2 Attacks
  - Two Time Pad
  - Birthday Attack
  - CBC Padding Attack
  - Length-extension on Merkle-Damgård
  - Timing attacks
  - Weaknesses in MSCHAPv2
- 3 HITCON Crypto Challenges
  - RSA with related messages
  - Combined Collision on short MD5/SHA1
- 4 Conclusion and Outlook

# Birthday Attack

- WEP: Assume every network card chooses an IV randomly.  
After how many packets do you get a first collision?
- Reminder: There are  $2^{24} \approx 16.7$  million IVs.

Your guesses?

# Birthday Attack

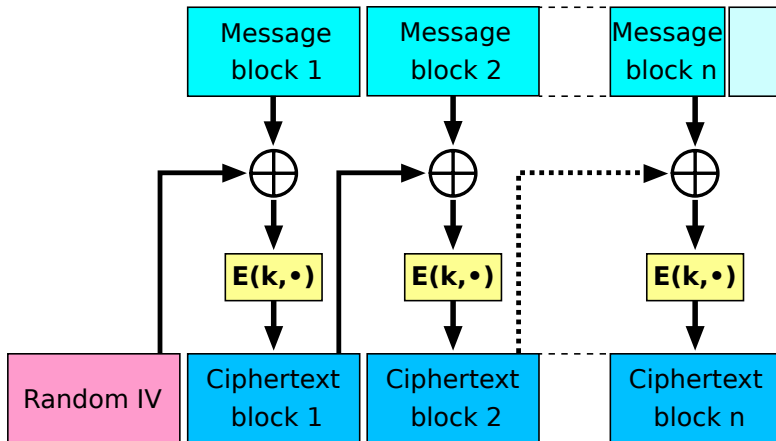
- Named after the birthday paradox: Given  $n$  people in a room, how large is the probability, that two people have the same birthday? Surprisingly large: 50% with 23 people, over 99% with 70.
- The birthday attack now just asks: What is the probability of a collision if you add  $n$  (“number of people”)  $k$ -length bitstrings (“number of days in a year”) to a set?
- Or different: How large (in average) must  $n$  be to get a collision? Answer:  $\approx 1.25\sqrt{2^k}$ .

# Outline

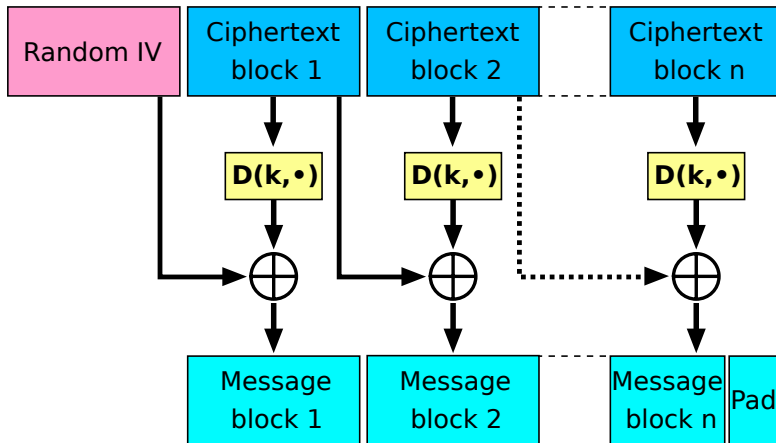
- 1 Crypto Basics
  - Cryptographic Primitives
  - XOR and the One Time Pad
- 2 Attacks
  - Two Time Pad
  - Birthday Attack
  - CBC Padding Attack
  - Length-extension on Merkle-Damgård
  - Timing attacks
  - Weaknesses in MSCHAPv2
- 3 HITCON Crypto Challenges
  - RSA with related messages
  - Combined Collision on short MD5/SHA1
- 4 Conclusion and Outlook



# CBC Mode Encryption



# CBC Mode Decryption



## CBC Padding Attack

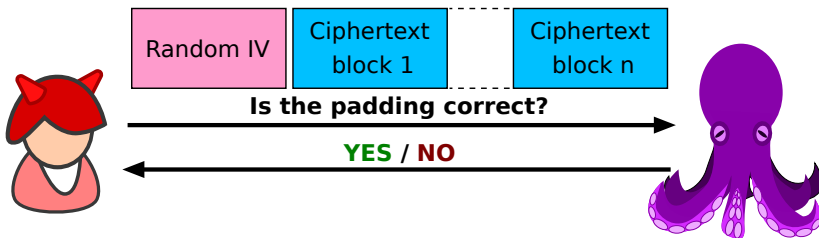
# Padding and the Padding Oracle

"Hi Bob": 

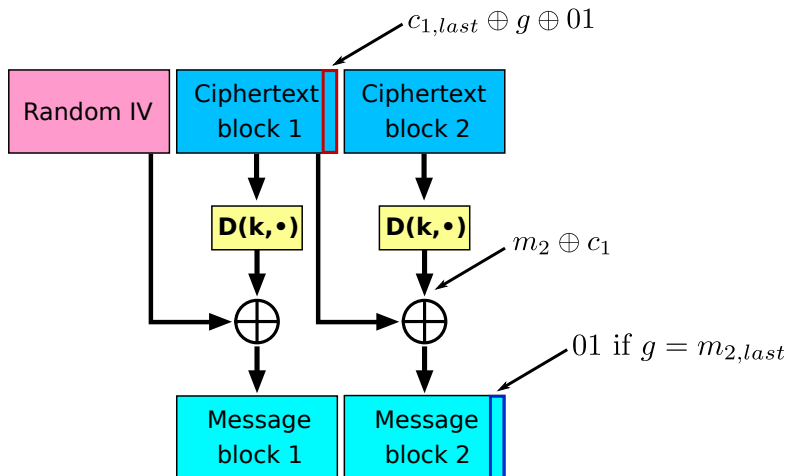
48	69	20	42	6f	62	02	02
----	----	----	----	----	----	----	----

"Hi Alice": 

48	69	20	41	6c	69	63	65
08	08	08	08	08	08	08	08



# CBC Padding Attack



Try to verify the output in the last message byte by applying the rules for XOR operations.

# CBC Padding Attack

Manipulate the last byte of ciphertext block 1 and try all possible 256 values of  $g$ . When you guessed the right  $g$  the padding oracle will say yes, otherwise (except in one special case) no.

The special case is simple and easy to check: Suppose your  $g$  produced a 02 at the end, but the payload before is also 02, in this case the padding is valid too. If the padding oracle says yes, just flip all bits in the byte before and check what the padding oracle says. When it still says yes, you produced 01 and  $g$  is right, otherwise you produced some other valid padding and your  $g$  is wrong.

Once you have the last byte guessed correctly, set it to  $c_1 \oplus g \oplus 02$  and continue to do the same thing for the byte before the last byte, but trying to produce 02 instead of 01 (and so on ...)

# CBC Padding Attacks in Real Life



Serge Vaudenay

*Security Flaws Induced by CBC Padding – Applications to SSL, IPSEC, WTLS.*

Proceedings of In Advances in Cryptology – EUROCRYPT 2002



Nadhem J. AlFardan, Kenneth G. Paterson

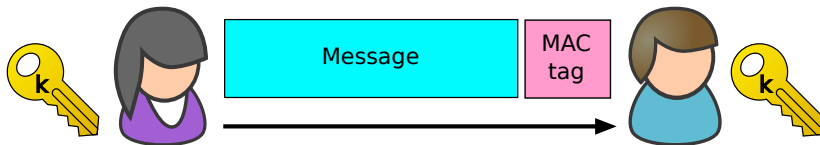
*Lucky Thirteen: Breaking the TLS and DTLS Record Protocols.*

Security and Privacy (SP), 2013 IEEE Symposium on (pp. 526-540), IEEE 2013

# Outline

- 1 Crypto Basics
  - Cryptographic Primitives
  - XOR and the One Time Pad
- 2 Attacks
  - Two Time Pad
  - Birthday Attack
  - CBC Padding Attack
  - Length-extension on Merkle-Damgård
  - Timing attacks
  - Weaknesses in MSCHAPv2
- 3 HITCON Crypto Challenges
  - RSA with related messages
  - Combined Collision on short MD5/SHA1
- 4 Conclusion and Outlook

# Insecure Message Authentication Code (MAC)



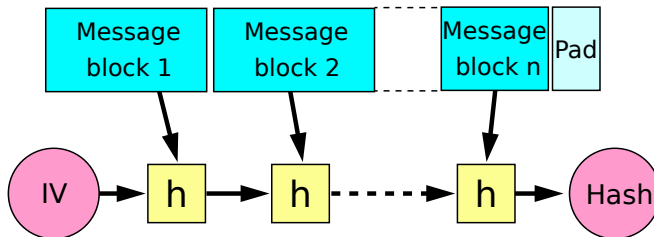
$$S(k, m) = \text{SHA256}(k \| m)$$

## Warning!

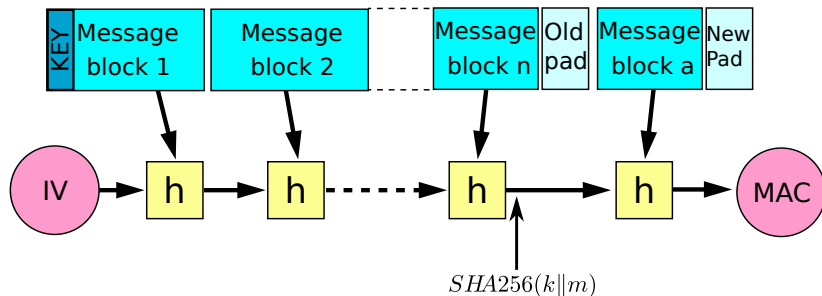
This Message Authentication Code is insecure!



# Merkle-Damgård Construction



# Length-extension against Merkle-Damgård Construction



Given the length of  $k||m$  and  $SHA256(k||m)$ , we can calculate  $SHA256(k||m||pad(k||m)||m_a)$  without knowing  $k$  and  $m$ . This is a valid MAC for  $m||pad(k||m)||m_a$ .

# Length-extension against Merkle-Damgård Construction

To calculate a valid MAC for  $m \parallel \text{pad}(k \parallel m) \parallel m_a$ , we first calculate  $p_0 = \text{pad}(k \parallel m)$ , which is a bit 1, followed by many bits 0, followed by the length of  $k \parallel m$  (64 bit big endian). You use as many bits 0 required to make the message length (including padding) the next multiple of the block size (512 bits).

We need another padding  $p_1$  with length of  $k \parallel m \parallel m_a$ .

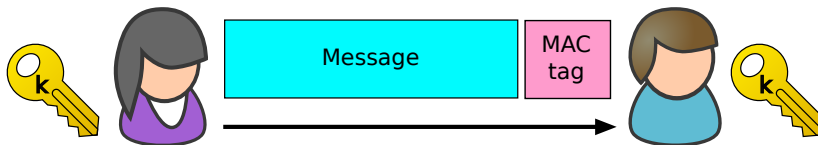
Now apply  $h$  to your new block with  $\text{SHA256}(k \parallel m)$  as previous output of  $h$  and  $m_a \parallel p_1$  as block. You may iterate  $h$  if it doesn't fit in a block.

Return the MAC as output of  $h$  together with the message  $m \parallel p_0 \parallel m_a$ .

# Outline

- 1 Crypto Basics
  - Cryptographic Primitives
  - XOR and the One Time Pad
- 2 Attacks
  - Two Time Pad
  - Birthday Attack
  - CBC Padding Attack
  - Length-extension on Merkle-Damgård
  - **Timing attacks**
  - Weaknesses in MSCHAPv2
- 3 HITCON Crypto Challenges
  - RSA with related messages
  - Combined Collision on short MD5/SHA1
- 4 Conclusion and Outlook

# A Secure Message Authentication Code



$$S(k, m) = \text{HMAC}_{\text{SHA256}}(k, m)$$

$$\text{HMAC}_H(k, m) = H(\text{opad} \oplus k \parallel H(\text{ipad} \oplus k \parallel m))$$

# Comparing MACs

## Vulnerable comparison of MACs

```
key = '(\x7f\x1(\xec[...] \xa0;\xf3\xa9'
message, given_mac = get_request()
if given_mac == hmac(sha256, key, message):
    process_message(message)
else:
    raise Exception('Invalid MAC!')
```

## Attention!

This code example is insecure!

The comparison stops once one byte differs. The idea of the timing attack is to measure this time and check how many bytes one got right. Longer time means more correct bytes (starting at the beginning)

# Comparing in constant time

## Constant time compare

```
def constant_time_compare(mac_a, mac_b):  
    if len(mac_a) != len(mac_b):  
        return False  
    result = 0  
    for a, b in zip(mac_a, mac_b):  
        result |= a ^ b  
    return result == 0
```

## Using defined function

```
if constant_time_compare(given_mac, correct_mac):  
    process_message(msg)
```

# Time Measuring Precision

Connection to system

Your guesses

Crosby et al.

---

Internet

20  $\mu$ s

LAN

100 ns



Scott. A. Crosby, Dan S. Wallach, Rudolf H. Riedi

*Opportunities and Limits of Remote Timing Attacks.*

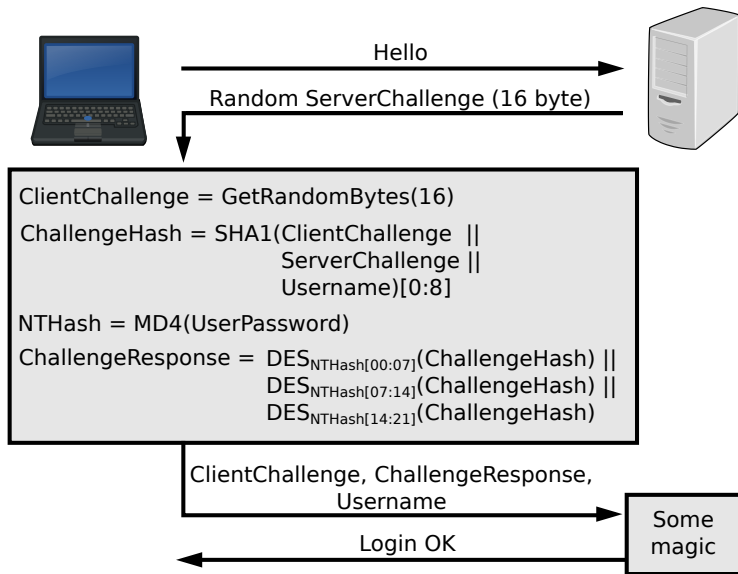
ACM Trans. Inf. Syst. Secur. 12, 3, Article 17, Jan. 2009



# Outline

- 1 Crypto Basics
  - Cryptographic Primitives
  - XOR and the One Time Pad
- 2 Attacks
  - Two Time Pad
  - Birthday Attack
  - CBC Padding Attack
  - Length-extension on Merkle-Damgård
  - Timing attacks
  - Weaknesses in MSCHAPv2
- 3 HITCON Crypto Challenges
  - RSA with related messages
  - Combined Collision on short MD5/SHA1
- 4 Conclusion and Outlook

# MSCHAPv2



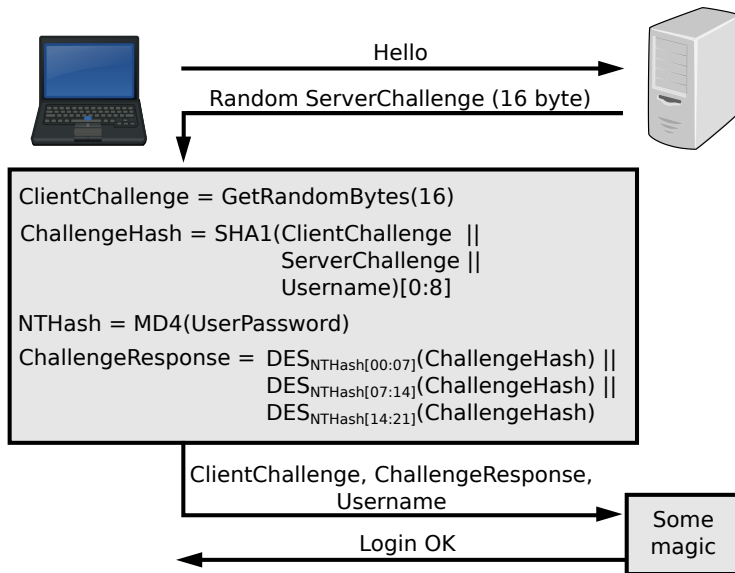
# Cracking DES

A keysize of 56 bit is vulnerable to exhaustive search.

History of breaking DES:

- 1998: Electronic Frontier Foundation built “Deep Crack” for \$250.000. Broke DES key in 56 hours.
- 2006: Universities of Bochum and Kiel built COPACOBANA for \$10.000. Needs 6.4 days to break a DES key.
- 2008: RIVYERA built by SciEngines can break DES key in less than one day on average.
- 2012: Cloudcracker offers breaking DES keys of MSCHAPv2 for \$200 in less than one day.

# MSCHAPv2 – Revisited



# MSCHAPv2 Weaknesses

The key of the last DES operation will be padded with null bytes, because MD4 is only 16 bytes, but we need 21 in total for the 3 DES keys. So there are only 2 bytes used in the third key thus  $2^{16}$  possible keys. Trying all keys takes less than a second on a normal PC.

The DES operations don't depend on each other. Therefore we have to search the whole keyspace (56 bit) only once and compare the output of the DES operation with the first and second part of the challenge hash (the third part was already broken, see above).

Once we have the keys for the DES operations we basically have the MD4 hash of the password, which is enough to authenticate.

# MSCHAPv2 Weaknesses

You can also speedup a dictionary attack by calculating the MD4 hashes of your words and put them into different files according to the last 2 bytes of the hash. Now break the last DES operation to get the last two bytes and try the words in the matching file. Instead of comparing  $n$  words, you just have to try  $\frac{n}{2^{16}}$  words.

# MSCHAPv2 – More weaknesses

There are even more weaknesses in MSCHAPv2 and related protocols, many of them are described in:



**Bruce Schneier, Mudge and David Wagner**

*Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2).*

Secure Networking–CQRE [Secure]'99. Springer Berlin Heidelberg, 1999. 192-203.

# Outline

- 1 Crypto Basics
  - Cryptographic Primitives
  - XOR and the One Time Pad
- 2 Attacks
  - Two Time Pad
  - Birthday Attack
  - CBC Padding Attack
  - Length-extension on Merkle-Damgård
  - Timing attacks
  - Weaknesses in MSCHAPv2
- 3 HITCON Crypto Challenges
  - RSA with related messages
  - Combined Collision on short MD5/SHA1
- 4 Conclusion and Outlook



# Outline

- 1 Crypto Basics
  - Cryptographic Primitives
  - XOR and the One Time Pad
- 2 Attacks
  - Two Time Pad
  - Birthday Attack
  - CBC Padding Attack
  - Length-extension on Merkle-Damgård
  - Timing attacks
  - Weaknesses in MSCHAPv2
- 3 HITCON Crypto Challenges
  - RSA with related messages
  - Combined Collision on short MD5/SHA1
- 4 Conclusion and Outlook

# Related background: RSA in a nutshell

## Key generation:

- ➊ Choose two primes  $p$  and  $q$  of approx. equal size
- ➋ Compute  $N = pq$  ("the modulus")
- ➌ Compute  $\varphi(N) = (p - 1)(q - 1)$
- ➍ Choose an integer  $e$  such that  $1 < e < \varphi(N)$  and  $e$  is coprime to  $\varphi(N)$
- ➎ Determine  $d$  such that  $ed \equiv 1 \pmod{\varphi(N)}$

Public key consists of  $e$  and  $N$ , the private key is  $d$  and  $N$ .  $p$ ,  $q$  and  $\varphi(N)$  are discarded (unless you use CRT)

**Encryption:**  $c = m^e \pmod{N}$

**Decryption:**  $m = c^d \pmod{N}$

**Attention!**

This is textbook RSA and not suitable for real encryption!

# The challenge

- Task: Decrypting RSA encrypted messages in 10 rounds.
- Each round the size of the modulus is increased.
- We get the ciphertexts  $c_1$  and  $c_2$  of two messages  $m_1$  and  $m_2 = m_1 + 1$ .  $m_1$  must be submitted.
- After submitting the 10th decrypted message, we get the encrypted flag (“the 11th round”)

```
def encrypt(bits, m):
    p = random_prime(bits)
    q = random_prime(bits)
    n = p * q
    assert m < n
    print n
    print m ** 3 % n
    print (m + 1) ** 3 % n
```

# RSA with related messages

- There is no message padding to destroy algebraic structure
- If two RSA messages have a known affine relation  $m_2 = \alpha m_1 + \beta$ , we can calculate  $m$  as fraction of two polynomials.
- For small  $e$  it's easy to find the polynomials.

Special case:  $e = 3$ :

$$\frac{\beta (c_2 + 2\alpha^3 c_1 - \beta^3)}{\alpha (c_2 - \alpha^3 c_1 + 2\beta^3)} = m_1 \pmod{N}$$

Plug in  $\alpha = \beta = 1$ ,  $c_1$  and  $c_2$  and you're done :-)



D. Coppersmith, M. Franklin, J. Patarin, and M. Reiter.  
*Low-Exponent RSA with Related Messages.*

Advances in Cryptology – EUROCRYPT'96 (pp. 1-9).

Springer Berlin Heidelberg.

# RSA with related messages: Implementation

Remember the formula: (with  $\alpha = \beta = 1$  and  $e = 3$ )

$$\frac{c_2 + 2c_1 - 1}{c_2 - c_1 + 2} = m_1 \pmod{N}$$

Put it into sage (<http://www.sagemath.org>):

```
n = 2004894888234189647743091889973
c1 = 1082466567248911881114327249964
c2 = 971721605305725141664027533790
m = ((c2 + 2*c1 - 1) * inverse_mod(c2-c1+2, n)) % n
print m # Output: 466926828657365800327978837323
```

Write a Python script which uses telnetlib and calls sage from command line to get  $m_1$ . Ugly, but works.

# More Attacks on RSA

If you are interested in more attacks on RSA, there is a good survey paper:



Dan Boneh

*Twenty years of attacks on the RSA cryptosystem.*

Notices of the AMS, Vol. 46, No. 2 (pp. 203-213) 1999

And some interesting side channel attacks:



Daniel Genkin, Adi Shamir, Eran Tromer

*RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis*

RSIACR Cryptology ePrint Archive 2013, 857



Daniel Genkin, Itamar Pipman, Eran Tromer

*Get Your Hands Off My Laptop: Physical Side-Channel Key-Extraction Attacks on PCs.*

Workshop on Cryptographic Hardware and ES 2014, to appear

# Outline

- 1 Crypto Basics
  - Cryptographic Primitives
  - XOR and the One Time Pad
- 2 Attacks
  - Two Time Pad
  - Birthday Attack
  - CBC Padding Attack
  - Length-extension on Merkle-Damgård
  - Timing attacks
  - Weaknesses in MSCHAPv2
- 3 HITCON Crypto Challenges
  - RSA with related messages
  - Combined Collision on short MD5/SHA1
- 4 Conclusion and Outlook

# Collision in wtf6: The Challenge

Note: I will only describe the crypto part of the challenge.

- Task: Find a collision on wtf6, both messages must end with "HITCON".
- wtf6(s) is the concatenation of the first 8 bytes of md5(s) and the last 8 bytes of sha1(s)
- Once you have a collision, your messages will be added bitwise modulo 256 and executed as Ruby code.



# Collision in wtf6: The execution

## Overview:

- ➊ Design Ruby code which ends with `\x00` and split it into two messages, which add up bitwise modulo 256.
- ➋ Generate a Chosen Prefix Collision in MD5 for these messages (see <https://code.google.com/p/hashclash>)
- ➌ Find a program which generates arbitrary MD5 collisions given an IV (`fastcoll`)
- ➍ Use Joux' method to generate about  $2^{32}$  MD5 multicollisions
- ➎ Use the multicollisions for a Birthday Attack on the last 8 bytes of SHA1

Steps 1-3 are considered solved. This is just Googling, compiling and executing programs.

# Joux' method

- Joux method allows us to build  $2^t$  multicollisions by just generating  $t$  hash collisions for given IVs.
- It assumes that your hash function is an iterated hash function (remember: Merkle-Damgård construction)



## Antoine Joux

*Multicollisions in iterated hash functions. Application to cascaded constructions..*

Advances in Cryptology – CRYPTO 2004 (pp. 306-316).  
Springer Berlin Heidelberg.

# Joux' method

## How it works:

Note: For simplicity we say “block”, but mean “message which length is a multiple of the block size”.

- ① Assume a Collision Machine  $C$ , which given an IV, returns two blocks  $b_0$  and  $b_1$ , such that:  $h(b_0, iv) = h(b_1, iv)$  and  $b_0 \neq b_1$
- ②  $iv_0 = h(\text{prefix})$   
 for  $i = 0 \dots t - 1$ :  
 $b_0, b_1 = C(iv_i)$   
 $M_i = \{b_0, b_1\}$   
 $iv_{i+1} = h(b_0, iv_i) = h(b_1, iv_i)$
- ③ Return  $M_0 \times M_1 \times \dots \times M_{t-1}$

# Joux' method: Python Code

```
def multicollision(t, iv=md5_iv):  
    blocks = []  
    last_h = iv  
    for i in range(t):  
        block0, block1 = collision_machine(last_h)  
        last_h = md5_raw(block0, last_h)  
        blocks.append((block0, block1))  
    for collision in itertools.product(*blocks):  
        yield b''.join(collision)
```

# Birthday attack on SHA1L8

## The unclever way:

Iterate over all multicollisions  $m$  and add  $SHA1L8(m_i || \text{"HITCON"})$  to a hashmap with value  $m_i$ . Abort and print messages when a collision is found.

## Drawbacks:

- $\approx 1.25 \cdot 2^{32}$  collisions to check in average (feasible)
- Assume 8 bytes each, we need 40 GiB of RAM to store just the hashes!

Some notes:

- $SHA1L8(m)$  denotes the last 8 bytes of  $SHA1(m)$
- If  $m_0, \dots, m_n$  are multicollisions, so are  $m_0 || \text{suffix}, \dots, m_n || \text{suffix}$

# Birthday Attack on SHA1L8

## The clever way:

- ➊ Iterate over all multicollisions  $m$  and calculate the hash  $h_i = \text{SHA1L8}(m_i || \text{"HITCON"})$ .
- ➋ Take the first byte of  $h_i$  as filename and append the last 7 bytes of  $h_i$  and 5 bytes for  $i$  to the file.
- ➌ Periodically load each file sequentially into memory and check for collision. Let  $i_0$  and  $i_1$  denote the  $i$  values of the collision.
- ➍ Calculate the two collision messages  $cm_0$  and  $cm_1$  by using  $i_0$  and  $i_1$ . (Just iterate the bits of  $i$  and either choose  $b_0$  or  $b_1$  depending on the bit, see Joux' method)
- ➎ Return  $cm_0 || \text{"HITCON"}$  and  $cm_1 || \text{"HITCON"}$

More optimizations: Use a graphic card, save prefixes of SHA1

# Birthday Attack on SHA1L8

## Some remarks on the clever way:

- This will still take some hours! And disk space! You can't fix that.
- If you assume  $1.25 \cdot 2^{32}$  multicollisions, it will take 60 GiB of disk space, but only 240+some MiB RAM.
- This is clearly possible :-)

# Outline

- 1 Crypto Basics
  - Cryptographic Primitives
  - XOR and the One Time Pad
- 2 Attacks
  - Two Time Pad
  - Birthday Attack
  - CBC Padding Attack
  - Length-extension on Merkle-Damgård
  - Timing attacks
  - Weaknesses in MSCHAPv2
- 3 HITCON Crypto Challenges
  - RSA with related messages
  - Combined Collision on short MD5/SHA1
- 4 Conclusion and Outlook



# Conclusion

- It's hard to get cryptography right
- Even small mistakes can render a crypto system useless
- Implementations can be insecure even if they are mathematically correct
- Good algorithms/implementations have been extensively reviewed

## Attention!

Never design cryptography yourself!

## Attention!

Never implement cryptography yourself!

# Outlook

- There are many more attacks I didn't cover
- Whole bunch of attacks on asymmetric cryptography. We need some number theory for that
- Some more classical things like “Meet in the Middle”, Bad Random Number Generators etc.
- Lots of techniques in cryptoanalysis (“differential cryptoanalysis”, “boomerang attack” etc.)
- If there is interest, I might do a sequel

# Literature and stuff

A good introductory book into cryptography:



Niels Ferguson, Bruce Schneier, Tadayoshi Kohno.

*Cryptography Engineering. Design Principles and Practical Applications.*

Wiley Publishing, 2010

A good, free MOOC (Massive Open Online Course):



Dan Boneh.

*Cryptography I.*

Stanford University at Coursera.

<https://www.coursera.org/course/crypto>

Much theory, but still good. Includes some practical exercises.

Next run: September, 8th for 6 weeks.