Zellic

intro2crypto

You had to be there for the attendance flag!

~RISC 30/7/25

Acknowledgment of Country

RISC acknowledges the people of the Woi Wurrung and Boon Wurrung language groups of the eastern Kulin Nation on whose unceded lands we conduct the business of the University and the club. RISC acknowledges their Ancestors and Elders, past, present, and emerging

Housekeeping

Housekeeping

Zellic

- You now have two weeks to solve challenges

Housekeeping

Zellic

- You now have two weeks to solve challenges
- Solutions are revealed in the next workshop

Housekeeping

Zellic

- You now have two weeks to solve challenges
- Solutions are revealed in the next workshop
  - Will also be on https://writeups.urisc.club

Housekeeping

Zellic

- You now have two weeks to solve challenges
- Solutions are revealed in the next workshop
  - Will also be on https://writeups.urisc.club
- Prizes! (Hopefully)

AI usage

Zellic

- We can’t stop you using AI

AI usage

Zellic

- We can’t stop you using AI
- But it doesn’t really teach you anything

AI usage

- We can’t stop you using AI
- But it doesn’t really teach you anything
- There’s a lot of satisfaction in reaching a solution by yourself :^)

AI usage

Zellic

- We can’t stop you using AI
- But it doesn’t really teach you anything
- There’s a lot of satisfaction in reaching a solution by yourself :^)
- Most modern CTF crypto challenges are beyond what any LLM can handle

AI usage

- We can’t stop you using AI
- But it doesn’t really teach you anything
- There’s a lot of satisfaction in reaching a solution by yourself :^)
- Most modern CTF crypto challenges are beyond what any LLM can handle
- You should learn to “think like an attacker”

AI usage

- We can’t stop you using AI
- But it doesn’t really teach you anything
- There’s a lot of satisfaction in reaching a solution by yourself :^)
- Most modern CTF crypto challenges are beyond what any LLM can handle
- You should learn to “think like an attacker”
- AI will also be utterly useless for future weeks

This week's sponsor

Zellic

Zellic

Zellic is a security research firm. Our targets include compilers, virtual machines, web apps, circuits, proof systems, and more. Before Zellic, we previously founded perfect blue, the #1 CTF team in 2020 and 2021. If you're smart and good at CTFs, we'd love to meet you.

We offer a complete benefits package and direct equity participation. We also offer flexible hours, remote work, and both full-time and part-time roles. Our team enjoys regular fully-funded offsites and range of other perks.

Ask your friends: you might already know someone who works here.

To learn more, check out our blog: zellic.io/auditooor-grindset

What is crypto?

Zellic

What is crypto?

- Imagine you’re passing notes in class.

What is crypto?

Zellic

- Imagine you're passing notes in class.

What is crypto?

Zellic

- Imagine you’re passing notes in class.
- Teacher catches you?

What is crypto?

- Imagine you’re passing notes in class.
- Teacher catches you?
- Little Jimmy snitches?

What is crypto?

Zellic

- Imagine you're passing notes in class.
- Teacher catches you?
- Little Jimmy snitches?
- How can we keep our very important messages away from prying eyes?

What is crypto?

Zellic

- Let’s think like a >2000 year old Roman dictator.

What is crypto?

- Let’s think like a >2000 year old Roman dictator.

What is crypto?

- Let's think like a >2000 year old Roman dictator.
- One approach is to shift letters by some amount.

What is crypto?

- Let's think like a >2000 year old Roman dictator.
- One approach is to shift letters by some amount.
  - ABCDEFGHIJKLMNOPQRSTUVWXYZ

    ↓ Shift by 13 letters

  - NOPQRSTUVWXYZABCDEFGHIJKLM

What is crypto?

Zellic

- Let’s think like a >2000 year old Roman dictator.
- One approach is to shift letters by some amount.
  - ABCDEFGHIJKLMNOPQRSTUVWXYZ

    ↓ Shift by 13 letters

  - NOPQRSTUVWXYZABCDEFGHIJKLM
- “u stink” -> “h fgvax"

Zellic

What is crypto?

- Let’s think like a >2000 year old Roman dictator.
- One approach is to shift letters by some amount.
  - `ABCDEFGHIJKLMNOPQRSTUVWXYZ`

    ↓ Shift by 6 letters

  - `GHIJKLMNOPQRSTUVWXYZABCDEF`
- “u stink” -> “a yzotq”

Zellic

What is crypto?

- Let’s think like a >2000 year old Roman dictator.
- One approach is to shift letters by some amount.
  - ABCDEFGHIJKLMNOPQRSTUVWXYZ

Shift by 6 letters

  - GHIJKLMNOPQRSTUVWXYZABCDEF
- “u stink” -> “a yzotq”
- Caesar Cipher

What is crypto?

- Let's think like a >200
- One approach is to sh
    - ABCDEFGHIJKLMN
    - GHIJKLMNOPQRSTU
- "u stink" -> "a yzotq"
- Caesar Cipher

What is crypto?

Zellic

- Obviously weak

What is crypto?

Zellic

- Obviously weak

h fgvax

i ghwby
j hixcz
k ijyda
l jkzeb
m klafc
n lmbgd
o mnche
p nodif
q opejg
r pqfkh
s qrgli
t rshmj
u stink
v tujol
w uvkpm
x vwlqn
y wxmro
z xynsp
a yzotq
b zapur
c abqvs
d bcrwt
e cdsxu
f detyv
g efuzw

What is crypto?

Zellic

- Obviously weak

h fgvax

i ghwby
j hixcz
k ijyda
l jkzeb
m klafc
n lmbgd
o mnche
p nodif
q opejg
r pqfkh
s qrgli
t rshmj
u stink
v tujol
w uvkpm
x vwlqn
y wxmro
z xynsp
a yzotq
b zapur
c abqvs
d bcrwt
e cdsxu
f detyv
g efuzw

What is crypto?

- Obviously weak
- Requires at most 25 brute force attempts

What is crypto?

- Obviously weak
- Requires at most 25 brute force attempts
- Computationally very cheap

What is crypto?

- Obviously weak
- Requires at most 25 brute force attempts
- Computationally very cheap
- https://gchq.github.io/CyberChef

Can we do better?

Can we do better?

Of course we can do better there's at least 50 slides left

Vigenère

- What if we rotated each letter by a different amount?

Vigenère

- What if we rotated each letter by a different amount?
  - Plaintext: "u stink"
  - Key: "ABC"

Zellic

Vigenère

- What if we rotated each letter by a different amount?
  - Plaintext: "u stink"
  - Key: "ABC"

0 1 2

Vigenère

Zellic

- What if we rotated each letter by a different amount?
  - Plaintext: "u stink"
  - Key: "ABC"
    - `0: ABCDEFGHIJKLMNOPQRSTUVWXYZ` (0)
    - `A: ABCDEFGHIJKLMNOPQRSTUVWXYZ` (0)
    - `B: BCDEFGHIJKLMNOPQRSTUVWXYZA` (1)
    - `C: CDEFGHIJKLMNOPQRSTUVWXYZAB` (2)

Vigenère

Zellic

- What if we rotated each letter by a different amount?
  - Plaintext: `"u stink"`
  - Key: `"ABCABC"`
    - `0: ABCDEFGHIJKLMNOPQRSTUVWXYZ` (0)
    - `A: ABCDEFGHIJKLMNOPQRSTUVWXYZ` (0)
    - `B: BCDEFGHIJKLMNOPQRSTUVWXYZA` (1)
    - `C: CDEFGHIJKLMNOPQRSTUVWXYZAB` (2)

Vigenère

- What if we rotated each letter by a different amount?
  - Plaintext: `"ustink"`
  - Key: `"ABCABC"`
    - `0: ABCDEFGHIJKLMNOPQRSTUVWXYZ` (0)
    - `A: ABCDEFGHIJKLMNOPQRSTUVWXYZ` (0)
    - `B: BCDEFGHIJKLMNOPQRSTUVWXYZA` (1)
    - `C: CDEFGHIJKLMNOPQRSTUVWXYZAB` (2)

Vigenère

Zellic

- What if we rotated each letter by a different amount?
  - Plaintext: `"ustink"`
  - Key: `"ABCABC"`

→ `utviom`

- `0: ABCDEFGHIJKLMNOPQRSTUVWXYZ` (0)
- `A: ABCDEFGHIJKLMNOPQRSTUVWXYZ` (0)
- `B: BCDEFGHIJKLMNOPQRSTUVWXYZA` (1)
- `C: CDEFGHIJKLMNOPQRSTUVWXYZAB` (2)

Vigenère

Zellic

- What if we rotated each letter by a different amount?
  - Plaintext: `"ustink"`
  - Key: `"ABCABC"` → `utviom`
    - 0: `ABCDEFGHIJKLMNOPQRSTUVWXYZ` (0)
    - A: `ABCDEFGHIJKLMNOPQRSTUVWXYZ` (0)
    - B: `BCDEFGHIJKLMNOPQRSTUVWXYZA` (1)
    - C: `CDEFGHIJKLMNOPQRSTUVWXYZAB` (2)

Vigenère

- But is this more secure?

Vigenère

- But is this more secure?
- We can't brute force...

Vigenère

- But is this more secure?
- We can't brute force...
- We would have to guess the whole key, right?

Vigenère

- But is this more secure?
- We can't brute force...
- We would have to guess the whole key, right?
- So it's secure!

Vigenère

Zellic

- But is this more secure?
- We can't brute force...
- We would have to guess the whole key, right?
- So it's secure!

Vigenère

- But is this more secure?
- We can't brute force...
- We would have to guess the whole key, right?
- So it's secure!
- Or is it...

Thought experiment

Thought experiment

If, for some Vigenère cipher:

- We know the key length is N
- We know N consecutive characters in the message

Can the cipher be broken? How?

Zellic

Vigenère

- What if we rotated each letter by a different amount?
  - Plaintext: tin
  - Key:

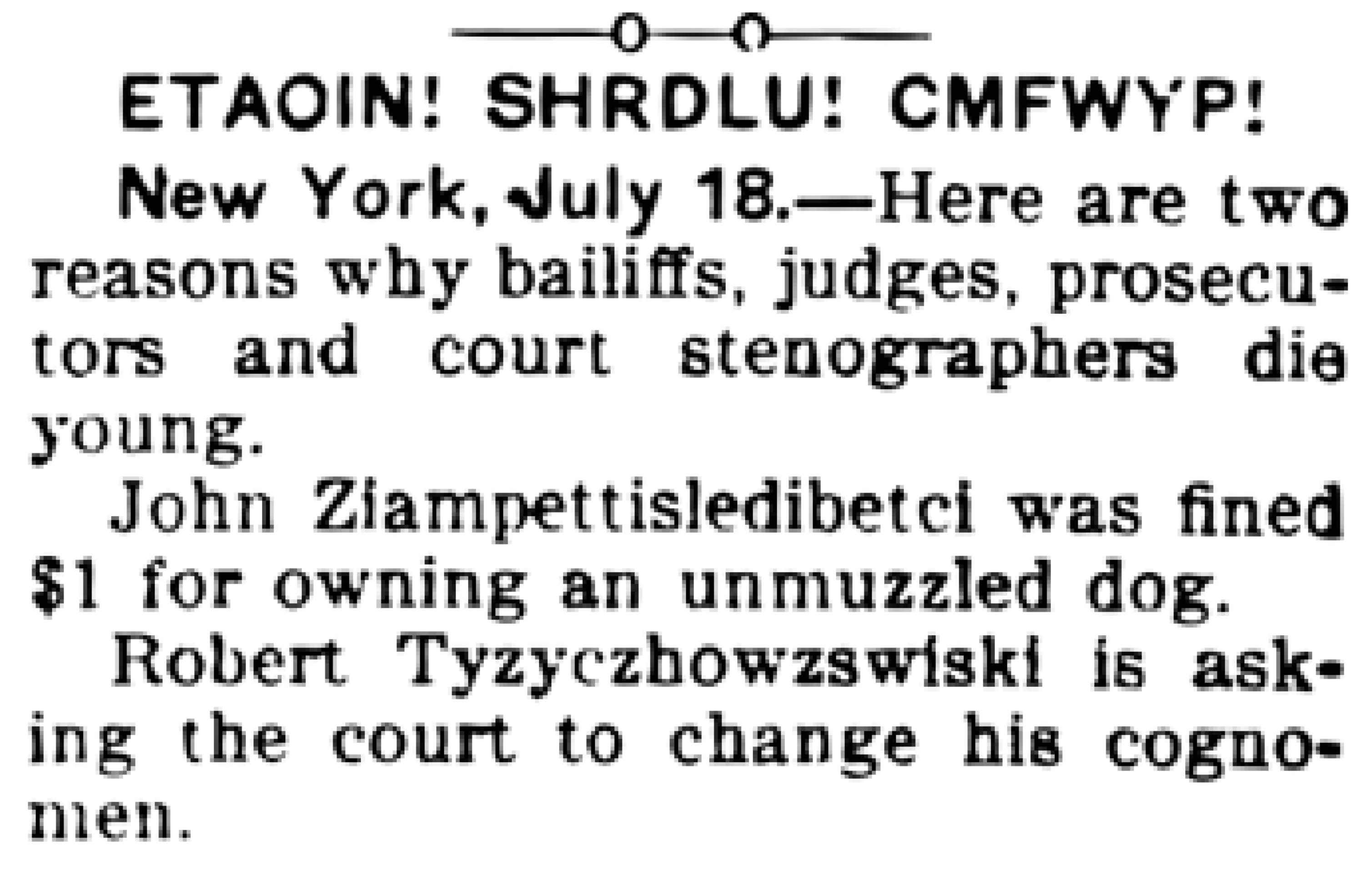
→ utviom

  - 0: ABCDEFGHIJKLMNOPQRSTUVWXYZ (0)

Vigenère

ETAOINSHRDLU

- All languages have a letter frequency distribution



ETAOIN! SHRDLU! CMFWYP!

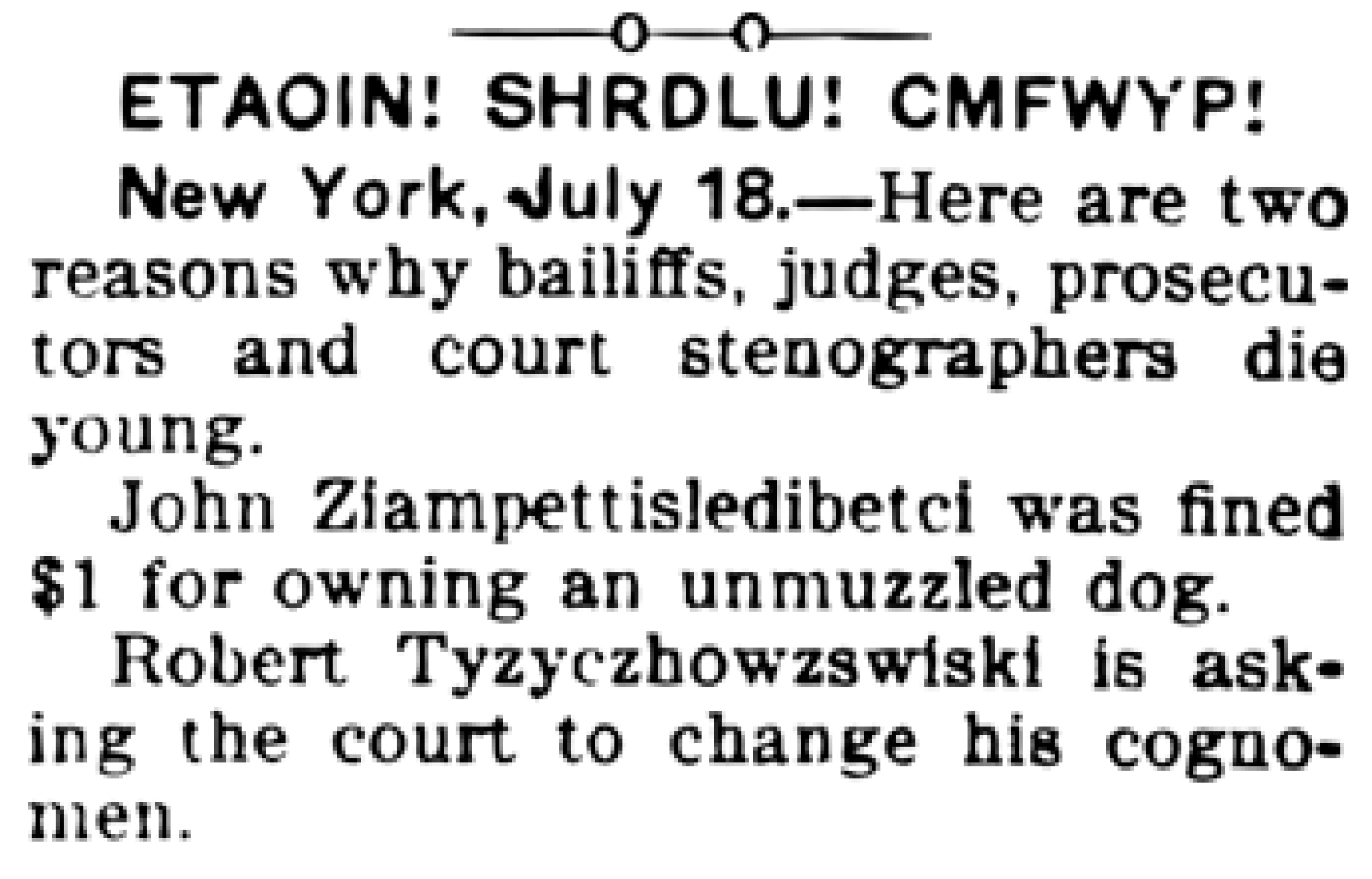
New York, July 18.—Here are two reasons why bailiffs, judges, prosecutors and court stenographers die young.

John Ziampettisledibetci was fined $1 for owning an unmuzzled dog.

Robert Tyzyczhowzswiski is asking the court to change his cognomen.

Vigenère
ETAOINSHRDLU

- All languages have a letter frequency distribution
- Let's consider the simpler case:
  - *monoalphabetic substitution cipher*



ETAOIN! SHRDLU! CMFWYP!

New York, July 18.—Here are two reasons why bailiffs, judges, prosecutors and court stenographers die young.

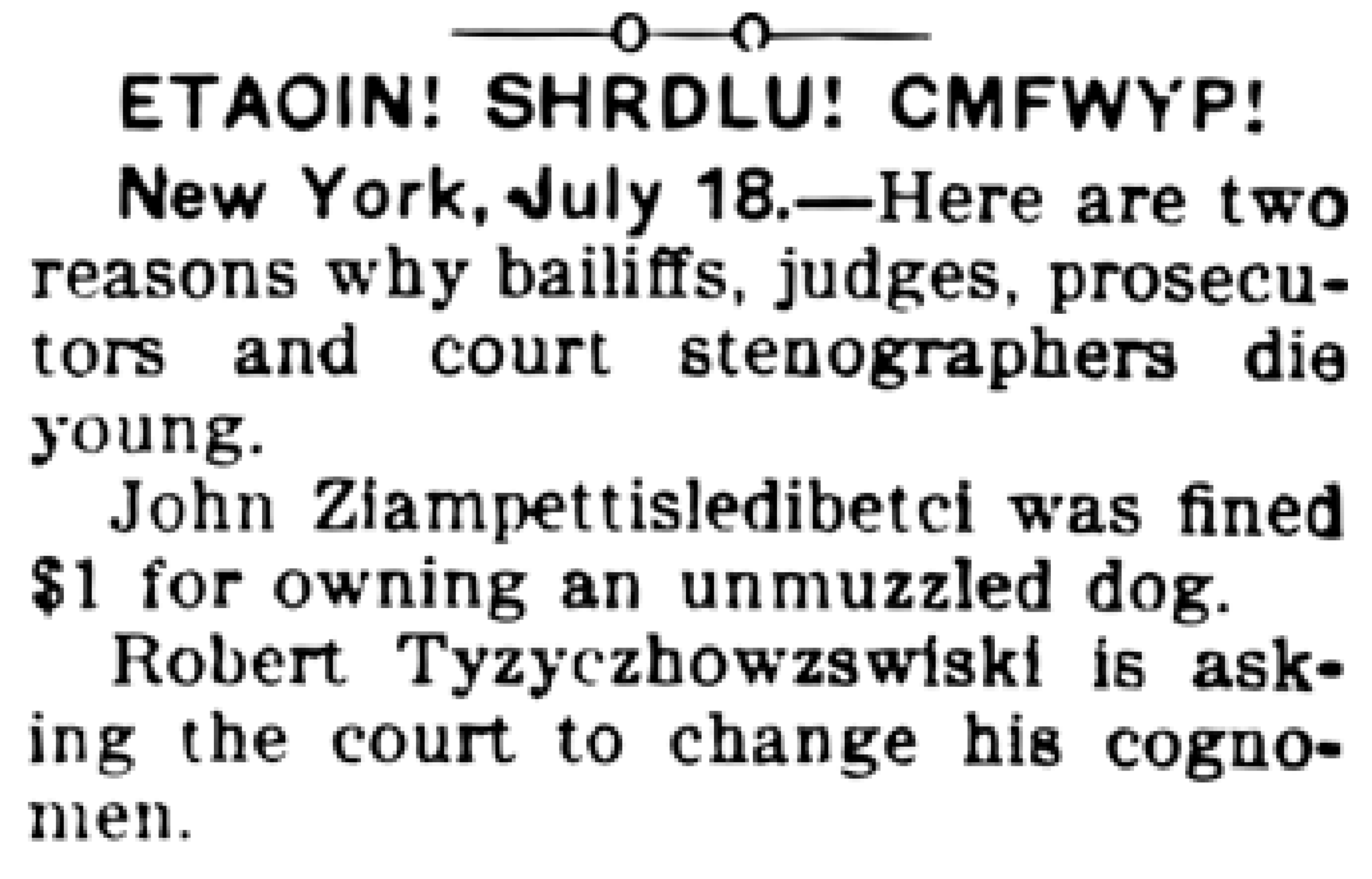
John Ziampettisledibetci was fined $1 for owning an unmuzzled dog.

Robert Tyzyczhowzswiski is asking the court to change his cognomen.

Zellic

Vigenère
ETAOINSHRDLU

- All languages have a letter frequency distribution
- Let's consider the simpler case:
    - *monoalphabetic substitution cipher*
    - A->X, B->D, C->W, D->J, ...



ETAOIN! SHRDLU! CMFWYP!

New York, July 18.—Here are two reasons why bailiffs, judges, prosecutors and court stenographers die young.

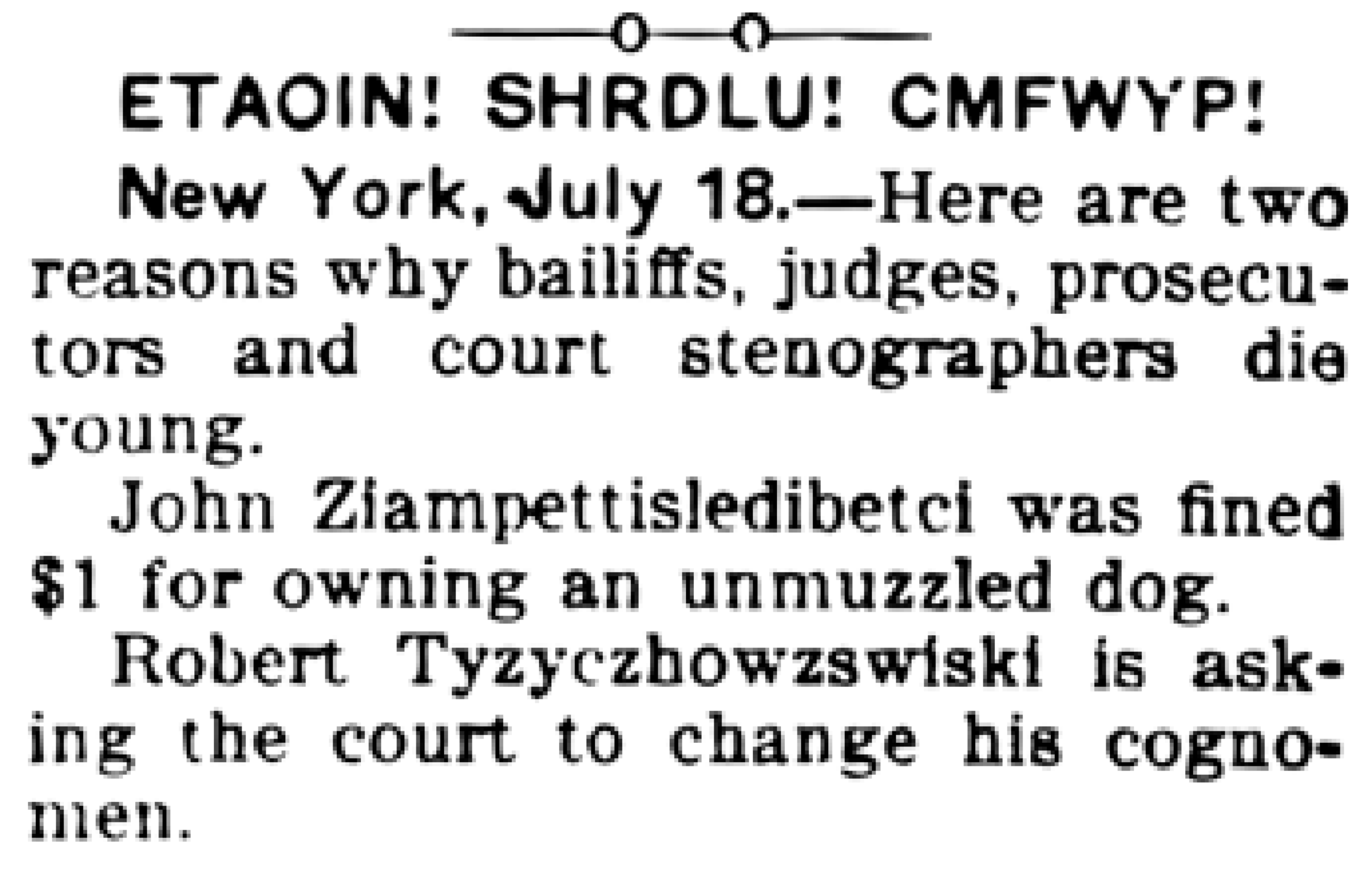
John Ziampettisledibetci was fined $1 for owning an unmuzzled dog.

Robert Tyzyczhowzswiski is asking the court to change his cognomen.

Vigenère

ETAOINSHRDLU

- All languages have a letter frequency distribution
- Let's consider the simpler case:
  - *monoalphabetic substitution cipher*
  - A->X, B->D, C->W, D->J, ...
- The most common letters in English are ETAOINSHRDLU (in order)



ETAOIN! SHRDLU! CMFWYP!

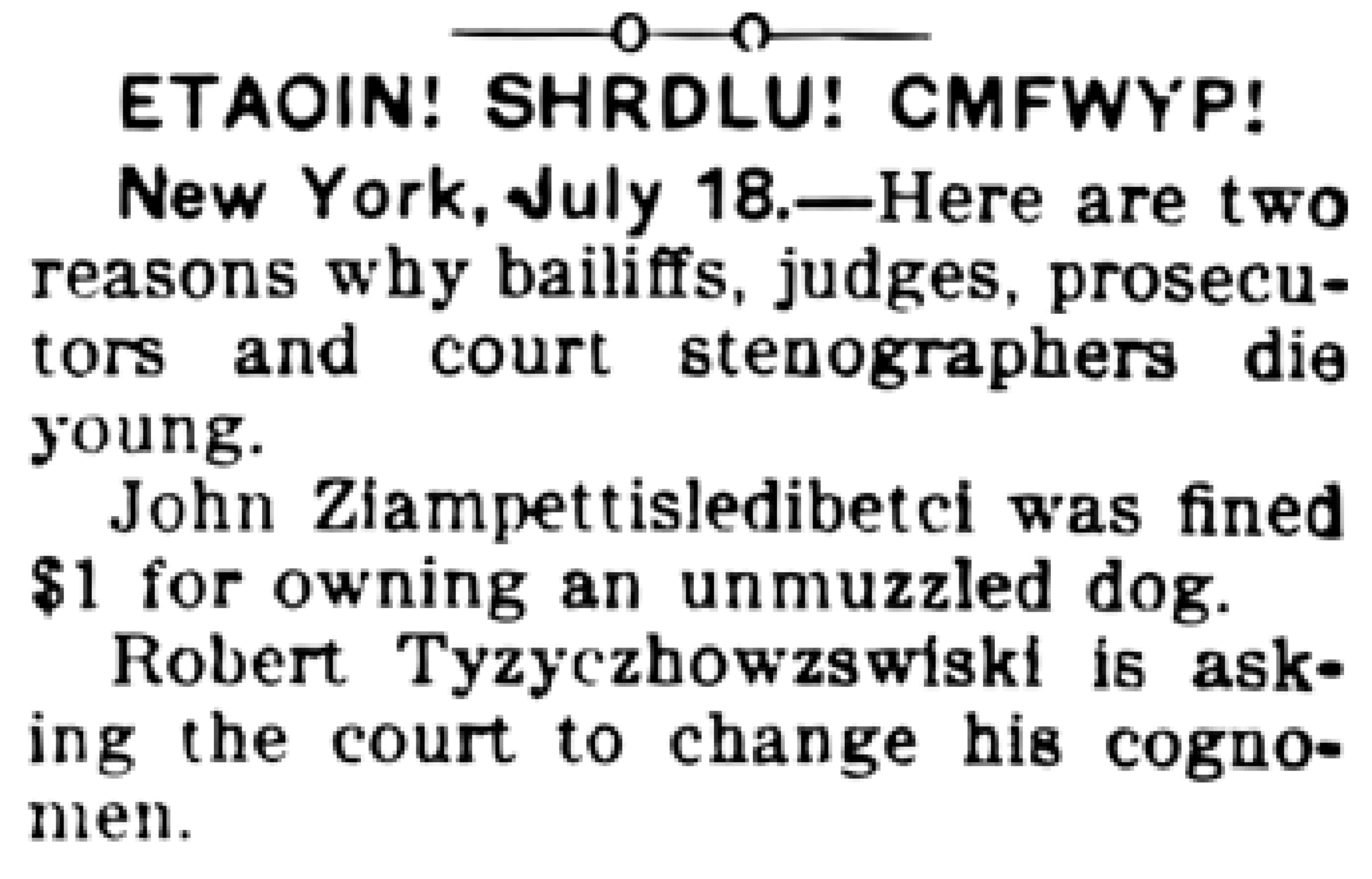
New York, July 18.—Here are two reasons why bailiffs, judges, prosecutors and court stenographers die young.

John Ziampettisledibetci was fined $1 for owning an unmuzzled dog.

Robert Tyzyczhowzswiski is asking the court to change his cognomen.

Vigenère
ETAOINSHRDLU

- All languages have a letter frequency distribution
- Let's consider the simpler case:
  - *monoalphabetic substitution cipher*
  - A->X, B->D, C->W, D->J, ...
- The most common letters in English are ETAOINSHRDLU (in order)
- The most frequent letters in the ciphertext are probably those in order too!


ETAOIN! SHRDLU! CMFWYP!

New York, July 18.—Here are two reasons why bailiffs, judges, prosecutors and court stenographers die young.

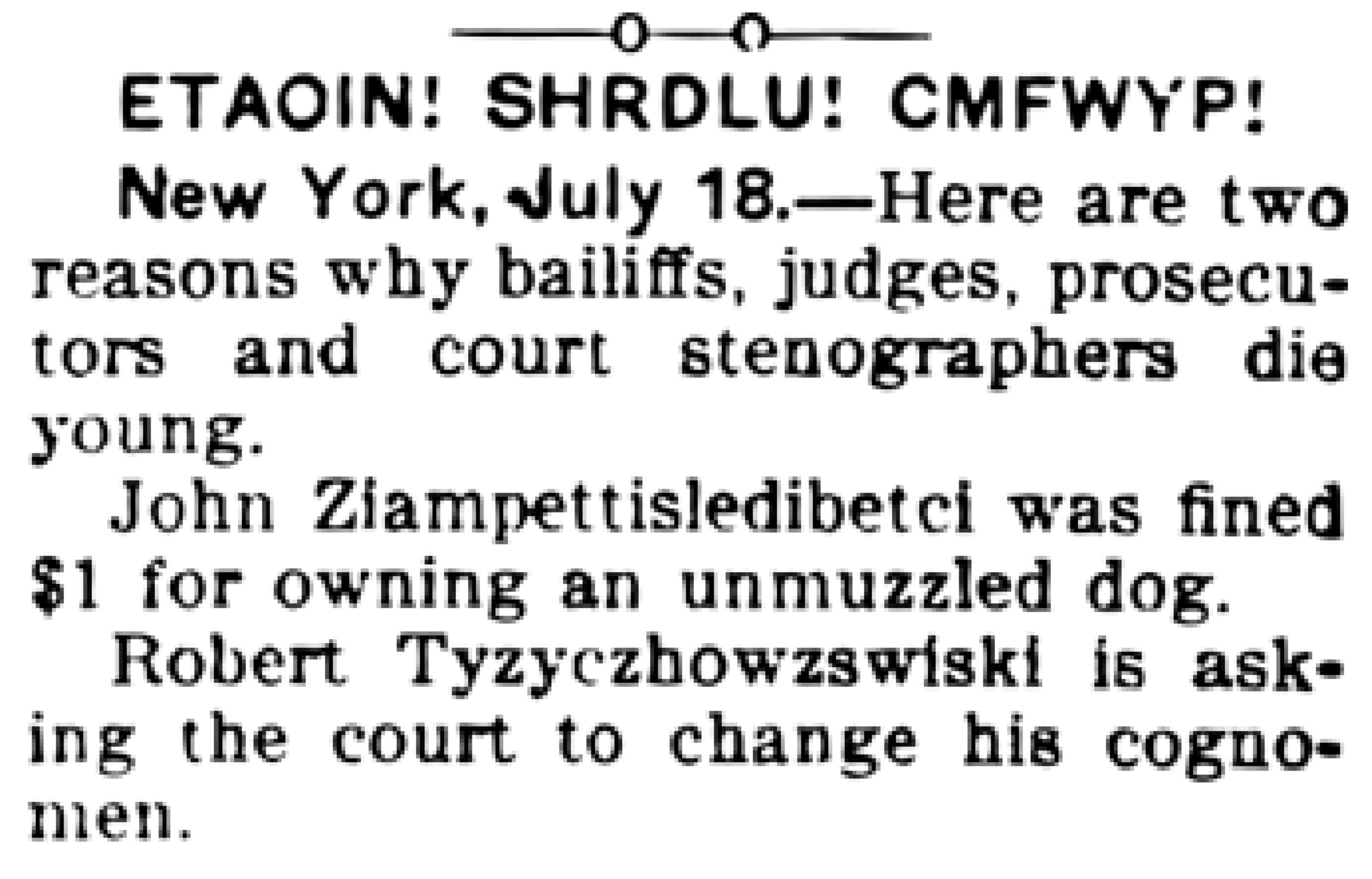
John Ziampettisledibetci was fined $1 for owning an unmuzzled dog.

Robert Tyzyczhowzswiski is asking the court to change his cognomen.

Vigenère

ETAOINSHRDLU

- All languages have a letter frequency distribution
- Let's consider the simpler case:
  - *monoalphabetic substitution cipher*
  - A->X, B->D, C->W, D->J, ...



ETAOIN! SHRDLU! CMFWYP!

New York, July 18.—Here are two reasons why bailiffs, judges, prosecutors and court stenographers die young.

John Ziampettisledibetci was fined $1 for owning an unmuzzled dog.

Robert Tyzyczhowzswiski is asking the court to change his cognomen.

- The most common letters in English are ETAOINSHRDLU (in order)
- The most frequent letters in the ciphertext are probably those in order too!
  - Fails with small messages, but with scale this becomes very precise

Zellic

Thought experiment #2

Thought experiment #2

If, for some Vigenère cipher:

- We know the length of the key
- The key is repeated some number of times
- The message is sufficiently long
- The message is in English

Can the cipher be broken? How?

Vigenère

Frequency Analysis with known key length

- Say we have a 4 letter key, *RISC*

Vigenère

Frequency Analysis with known key length

- Say we have a 4 letter key, *RISC*
- We can rearrange our ciphertext into 4 columns

Vigenère

Frequency Analysis with known key length

- Say we have a 4 letter key, *RISC*
- We can rearrange our ciphertext into 4 columns
  - 0, 4, 8, 12, 16, …
  - 1, 5, 9, 13, 17, …
  - 2, 6, 10, 14, 18, …
  - 3, 7, 11, 15, 19, …

Vigenère

Frequency Analysis with known key length

- Say we have a 4 letter key, *RISC*
- We can rearrange our ciphertext into 4 columns
  - 0, 4, 8, 12, 16, …
  - 1, 5, 9, 13, 17, …
  - 2, 6, 10, 14, 18, …
  - 3, 7, 11, 15, 19, …

R

Vigenère

Frequency Analysis with known key length

- Say we have a 4 letter key, *RISC*
- We can rearrange our ciphertext into 4 columns
  - 0, 4, 8, 12, 16, …
  - 1, 5, 9, 13, 17, …
  - 2, 6, 10, 14, 18, …
  - 3, 7, 11, 15, 19, …

Zellic

Vigenère

Frequency Analysis with known key length

- Say we have a 4 letter key, *RISC*
- We can rearrange our ciphertext into 4 columns
  - 0, 4, 8, 12, 16, …
  - 1, 5, 9, 13, 17, …
  - 2, 6, 10, 14, 18, … ← S
  - 3, 7, 11, 15, 19, …

Zellic

Vigenère

Frequency Analysis with known key length

- Say we have a 4 letter key, *RISC*
- We can rearrange our ciphertext into 4 columns
  - 0, 4, 8, 12, 16, …
  - 1, 5, 9, 13, 17, …
  - 2, 6, 10, 14, 18, …
  - 3, 7, 11, 15, 19, … ← C

Vigenère

Frequency Analysis with known key length

- Say we have a 4 letter key, *RISC*
- We can rearrange our ciphertext into 4 columns
- Each column represents part of the message that was encrypted with 1 letter

Vigenère

Frequency Analysis with known key length

- Say we have a 4 letter key, *RISC*
- We can rearrange our ciphertext into 4 columns
- Each column represents part of the message that was encrypted with 1 letter
  - 4 Caesar ciphers

Vigenère

Frequency Analysis with known key length

- Say we have a 4 letter key, *RISC*
- We can rearrange our ciphertext into 4 columns
- Each column represents part of the message that was encrypted with 1 letter
  - 4 Caesar ciphers
- Brute forcing is still hard, but we can use frequency analysis

Vigenère

Frequency Analysis with known key length

- Say we have a 4 letter key, *RISC*
- We can rearrange our ciphertext into 4 columns
- Each column represents part of the message that was encrypted with 1 letter
  - 4 Caesar ciphers
- Brute forcing is still hard, but we can use frequency analysis
- Reduces attack from 26^N to 26*N for key of length N

Vigenère

Frequency Analysis with known key length

- Say we have a 4 letter key, *RISC*
- We can rearrange our ciphertext into 4 columns
- Each column represents part of the message that was encrypted with 1 letter
  - 4 Caesar ciphers
- Brute forcing is still hard, but we can use frequency analysis
- Reduces attack from 26^N to 26*N for key of length N
  - In this case, 456,976 to 104

Thought experiment #3

Thought experiment #3

If, for some Vigenère cipher:

- The key is longer than the message
- The key is truly random
- The key is never reused

Can the cipher still be broken? How?

Vigenère

OTP

- If the key is longer than the message

Vigenère
OTP

- If the key is longer than the message
- And the key is never used more than once

Vigenère
OTP

- If the key is longer than the message
- And the key is never used more than once
- Perfect secrecy has been achieved!

Vigenère
OTP

- If the key is longer than the message
- And the key is never used more than once
- Perfect secrecy has been achieved!
- Even if some of the plaintext is known (say, you always start with “Dear X”)

Vigenère

OTP

- If the key is longer than the message
- And the key is never used more than once
- Perfect secrecy has been achieved!
- Even if some of the plaintext is known (say, you always start with “Dear X”)
- No information about the rest of the message is recovered

Vigenère
OTP

- If the key is longer than the message
- And the key is never used more than once
- Perfect secrecy has been achieved!
- Even if some of the plaintext is known (say, you always start with “Dear X”)
- No information about the rest of the message is recovered
- This was actually how cold-war spies received orders from HQ!

Vigenère
OTP

- If the key is longer than the message
- And the key is never used more than once
- Perfect secrecy has been achieved!
- Even if some of the plaintext is known (say, you always start with “Dear X”)
- No information about the rest of the message is recovered
- This was actually how cold-war spies received orders from HQ!
    - Further reading: Number Stations

Vigenère
OTP

- If the key is longer than the message
- And the key is never used more than once
- Perfect secrecy has been achieved!
- Even if some of the plaintext is known (say, you always start with “Dear X”)
- No information about the rest of the message is recovered
- This was actually how cold-war spies received orders from HQ!
    - Further reading: Number Stations

ALLEGEDLY

Vigenère
OTP

- If the key is longer than the message
- And the key is never used more than once
- Perfect secrecy has been achieved!
- Even if some of the plaintext is known (say, you always start with “Dear X”)
- No information about the rest of the message is recovered
- This was actually how cold-war spies received orders from HQ!
    - Further reading: Number Stations

Probably still in use today by some countries, but this is pure speculation

Zellic

But how do you exchange the key?

XOR

Winding forward ~500 years

XOR

Winding forward ~500 years

- Encrypting only letters isn't super useful

XOR

Winding forward ~500 years

- Encrypting only letters isn't super useful
- Maybe we want:
  - Numbers

XOR

Winding forward ~500 years

- Encrypting only letters isn't super useful
- Maybe we want:
  - Numbers
  - Uppercase/lowercase

XOR

Winding forward ~500 years

- Encrypting only letters isn't super useful
- Maybe we want:
  - Numbers
  - Uppercase/lowercase
  - Special characters

XOR

Winding forward ~500 years

- Encrypting only letters isn’t super useful
- Maybe we want:
  - Numbers
  - Uppercase/lowercase
  - Special characters
  - The full range of a byte? (0-255)

XOR

Winding forward ~500 years

- Bitwise operation

XOR

Winding forward ~500 years

- Bitwise operation
- Super fast on modern CPUs (ancient ones too)

XOR

Winding forward ~500 years

- Bitwise operation
- Super fast on modern CPUs (ancient ones too)
  - 1 clock cycle

XOR

Winding forward ~500 years

- Bitwise operation
- Super fast on modern CPUs (ancient ones too)
  - 1 clock cycle
- Properties that make it useful for crypto

XOR

Winding forward ~500 years

- Bitwise operation
- Super fast on modern CPUs (ancient ones too)
  - 1 clock cycle
- Properties that make it useful for crypto

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Zellic

XOR

Winding forward ~500 years

- Bitwise operation
- Super fast on modern CPUs (ancient ones too)
  - 1 clock cycle
- Properties that make it useful for crypto
  - Balanced outputs (AND has 3 0's, OR has 3 1's)

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

XOR

Involution

- XOR is it’s own inverse

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

XOR

Involution

- XOR is it’s own inverse
- $(A \oplus B) \oplus B = A$

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

XOR

Involution

- XOR is it's own inverse
- $(A \oplus B) \oplus B = A$
  - $A \oplus (B \oplus B) = A$

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Zellic

XOR

Involution

- XOR is it's own inverse
- $(A \oplus B) \oplus B = A$
  - $A \oplus (B \oplus B) = A$
  - $A \oplus () = A$

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

XOR

Involution

- XOR is it’s own inverse
- $(A \oplus B) \oplus B = A$
- $A \oplus B = C \Rightarrow A = B \oplus C$

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

XOR

Involution

- XOR is it's own inverse
- $(A \oplus B) \oplus B = A$
- $A \oplus B = C \Rightarrow A = B \oplus C$
  - $A \oplus B \oplus B = C \oplus B$

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

XOR

Involution

- XOR is it’s own inverse
- $(A \oplus B) \oplus B = A$
- $A \oplus B = C \Rightarrow A = B \oplus C$
  - $A = C \oplus B$

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

XOR

Involution

- XOR is it's own inverse
- $(A \oplus B) \oplus B = A$
- $A \oplus B = C \Rightarrow A = B \oplus C$

- NB: We use $\oplus$ to represent XOR in slides
  - You will see ^ used in code to represent the same operation

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

XOR

Cryptography

- If we have some message M

XOR

Cryptography

- If we have some message M
- And some key K

XOR

Cryptography

- If we have some message M
- And some key K
- We can obtain ciphertext $C = M \oplus K$

XOR

Cryptography

- If we have some message M
- And some key K
- We can obtain ciphertext $C = M \oplus K$
- And we can decrypt with $M = C \oplus K$

XOR

Cryptography

- If we have some message M
- And some key K
- We can obtain ciphertext $C = M \oplus K$
- And we can decrypt with $M = C \oplus K$
- Question: Do you notice any similarities to Vigenère here? Do the same attacks work?

Thought experiment #4

Thought experiment #4

How is XOR encryption similar to Vigenère?

Do the same attacks work?

What new attacks emerge?

XOR

- Same problems as Vigenère

XOR

Zellic

- Same problems as Vigenère
  - Column frequency analysis

XOR

- Same problems as Vigenère
  - Column frequency analysis
  - Small key brute force

XOR

Zellic

- Same problems as Vigenère
  - Column frequency analysis
  - Small key brute force
  - Key distribution

XOR

- Same problems as Vigenère
  - Column frequency analysis
  - Small key brute force
  - Key distribution

XOR

Zellic

- Same problems as Vigenère
  - Column frequency analysis
  - Small key brute force
  - Key distribution (unless you're a cold war era spy)

RSA

- Finding factors of numbers isn't really fun

RSA

- Finding factors of numbers isn't really fun
- In fact, it's not fun for computers either

RSA

Zellic

- Finding factors of numbers isn’t really fun
- In fact, it’s not fun for computers either
- While 64 bit numbers can be factored incredibly fast…

RSA

Zellic

- Finding factors of numbers isn’t really fun
- In fact, it’s not fun for computers either
- While 64 bit numbers can be factored incredibly fast…
- … what about 2048 bits? 4096? 8192?

RSA

- Finding factors of numbers isn’t really fun
- In fact, it’s not fun for computers either
- While 64 bit numbers can be factored incredibly fast…
- … what about 2048 bits? 4096? 8192?
- It get’s even worse if the number is the product of two N bit primes!

RSA

- Finding factors of numbers isn’t really fun
- In fact, it’s not fun for computers either
- While 64 bit numbers can be factored incredibly fast…
- … what about 2048 bits? 4096? 8192?
- It get’s even worse if the number is the product of two N bit primes!
  - Can’t divide by 2, 3, 5, 7, …

RSA

Integer Factorisation

- Unsolved problem - “Can an integer be factored in polynomial time”

RSA
Integer Factorisation

*on a classical computer

- Unsolved problem - “Can an integer be factored in polynomial time”

RSA

Integer Factorisation

*on a classical computer

- Unsolved problem - “Can an integer be factored in polynomial time”
- Generally assumed to be NP
  - Hard to find a solution

RSA
Integer Factorisation

*on a classical computer

- Unsolved problem - “Can an integer be factored in polynomial time”
- Generally assumed to be NP
  - Hard to find a solution
  - Easy to verify a solution

RSA
Integer Factorisation

*on a classical computer

- Unsolved problem - “Can an integer be factored in polynomial time”
- Generally assumed to be NP
  - Hard to find a solution (hard to find factors of a given number)
  - Easy to verify a solution (easy to check if A and B are factors of a number)

RSA
Integer Factorisation

*on a classical computer

- Unsolved problem - “Can an integer be factored in polynomial time”
- Generally assumed to be NP
  - Hard to find a solution   (hard to find factors of a given number)
  - Easy to verify a solution (easy to check if A and B are factors of a number)

- If you disagree…

RSA

Integer Factorisation

*on a classical computer

- Unsolved proble… al time”
- Generally assur…
  - Hard to find a… mber)
  - Easy to verify… ors of a number)
- If you disagree.…


```
cb 12 fd 3b 32 8c 65 17 ff 39 2f 25 27 e3 80 ba
bf d7 e4 5f 9a 65 a9 96 70 96 ef f9 49 36 79 97
e4 22 23 4c 9d af 5b 27 56 ef 6a 36 3f 4a 5d d1
44 fb 5d ca 21 7a f3 7c 39 cb ab 07 1c 6a ec 2c
21 64 37 1d 16 11 73 3f 7e 1f 68 a9 ea b5 bd 7a
05 6d 38 05 8d ef ee 23 1c e2 cf ec aa 22 d9 4e
84 47 38 c2 cd bc 1b 72 51 a3 64 46 f0 55 95 57
ee de 87 db 39 96 57 c0 42 58 1b 48 bc 5c 79 20
d9 96 4e e9 49 86 67 78 4f fe 4b 66 b0 f6 7d b9
e7 07 de c6 da d8 20 96 65 a0 de 4e a0 c4 76 f3
41 e7 e4 de c0 32 47 8d 5f a9 96 09 b8 46 5e e8
c0 3e d1 d0 69 e8 4c 26 3c 8e 69 1c 01 eb 61 ec
ec 77 f0 e9 c2 fe 2a bf 8d 68 c2 1a 55 7d 61 ac
85 c8 f7 16 e2 a0 73 97 ff 26 5c 05 38 e6 e1 a7
89 13 d6 ac 13 aa 7e 44 87 83 07 ab f2 da a6 cf
38 a7 6b cb 17 07 62 08 a9 10 8e 58 8d 73 c6 e9
```


^Bank of America’s public key.

RSA

Integer Factorisation

- RSA’s security relies on the fact that factoring is hard

RSA

Integer Factorisation

- RSA’s security relies on the fact that factoring is hard
- Current record is factoring a 795-bit number on specialised hardware

RSA

Integer Factorisation

- RSA’s security relies on the fact that factoring is hard
- Current record is factoring a 795-bit number on specialised hardware
  - A lot of smaller numbers (<128 bit) have known factors on FactorDB.com

RSA
Integer Factorisation

- RSA's security relies on the fact that factoring is hard
- Current record is factoring a 795-bit number on specialised hardware
    - A lot of smaller numbers (<128 bit) have known factors on FactorDB.com
- How do we go from factoring -> encryption?

The next bit is math heavy

(sorry)

RSA

Modular Arithmetic - Intuition

- It’s 18:00 right now. What time will it be in 219 hours?

RSA

Modular Arithmetic - Intuition

- It's 18:00 right now. What time will it be in 219 hours?
- Hard way:
  - Add 6 hours -> midnight, 13 hours left
  - Add 12 hours -> midday, 1 hour left
  - … repeat many, many, many times
  - Add remainder -> 9PM / 21:00
  - Answer: 9PM / 21:00

RSA

Modular Arithmetic - Intuition

- It's 18:00 right now. What time will it be in 219 hours?
- Easy way:
  - 18 + 219 -> 237
  - We want an answer in [0,24), so divide by 24 and get the remainder
  - 237 / 24 = 9 r21
  - Answer: 9PM / 21:00

RSA

Modular Arithmetic

- System of arithmetic where values wrap around after a certain value (modulus)

RSA

Modular Arithmetic

- System of arithmetic where values wrap around after a certain value (modulus)
  - In our time example, the modulus would be 24 (or 12 for AM/PM format)

RSA

Modular Arithmetic

- System of arithmetic where values wrap around after a certain value (modulus)
  - In our time example, the modulus would be 24 (or 12 for AM/PM format)

- Alternatively: “Remainder of division”
  - 237 mod 24 → 237 / 24 = 9 r21
  - 237 mod 24 ≡ 21

RSA

Modular Arithmetic

- Only defined for the integers, $\mathbb{Z}$

RSA

Modular Arithmetic

- Only defined for the integers, $\mathbb{Z}$
- Inherits associativity, commutativity, distributivity, …

RSA

Modular Arithmetic

- Only defined for the integers, $\mathbb{Z}$
- Inherits associativity, commutativity, distributivity, …
  - $(a + b) \bmod n \Leftrightarrow (b + a) \bmod n$

RSA

Modular Arithmetic

- Only defined for the integers, $\mathbb{Z}$
- Inherits associativity, commutativity, distributivity, …
  - (a + b) mod n ⇔ (b + a) mod n
  - a mod n * b mod n ⇔ (a * b) mod n

RSA

Modular Arithmetic

- Only defined for the integers, $\mathbb{Z}$
- Inherits associativity, commutativity, distributivity, …
  - $(a + b) \bmod n \Leftrightarrow (b + a) \bmod n$
  - $a \bmod n * b \bmod n \Leftrightarrow (a * b) \bmod n$
- Division is not defined

RSA

Asleep yet?

- $8 + 11 \mod 13 \equiv ?$
- $9 * 8 \mod 11 \equiv ?$
- $28{,}472 \mod 1{,}824{,}792 \equiv ?$

RSA

Asleep yet?

- $8 + 11 \bmod 13 \equiv 6$
- $9 * 8 \bmod 11 \equiv 6$
- $28{,}472 \bmod 1{,}824{,}792 \equiv 28{,}472$

RSA

Asleep yet?

- 8 + 11 mod 13 ≡ 6
- 9 * 8 mod 11 ≡ 6
- 28,472 mod 1,824,792 ≡ 28,472

8 + 11 = 19
19 mod 13 ≡ 6
19/13 = 1 r6

RSA

Asleep yet?

- $8 + 11 \bmod 13 \equiv 6$
- $9 * 8 \bmod 11 \equiv 6$
- $28{,}472 \bmod 1{,}824{,}792 \equiv 28{,}472$

$9 * 8 = 72$

$72 \bmod 11 \equiv 6$

$72/11 = 6 \text{ r}6$

RSA

Asleep yet?

- 8 + 11 mod 13 ≡ 6
- 9 * 8 mod 11 ≡ 6
- 28,472 mod 1,824,792 ≡ 28,472

28,472 mod 1,824,792 ≡ 28,472
28,472/1,824,792 = 0 r28,472

RSA

Modular Multiplicative Inverse

- In normal math, $x^{-1}x = 1$
  - Since $x^{-1}x \Leftrightarrow x/x$

RSA

Modular Multiplicative Inverse

- In normal math, $x^{-1}x = 1$
  - Since $x^{-1}x \Leftrightarrow x/x$
- This also holds in modular arithmetic, but…

RSA

Modular Multiplicative Inverse

- In normal math, $x^{-1}x = 1$
  - Since $x^{-1}x \Leftrightarrow x/x$
- This also holds in modular arithmetic, but…
- What is x^{-1} in modular arithmetic?

RSA

Modular Multiplicative Inverse

- In normal math, $x^{-1}x = 1$
  - Since $x^{-1}x \Leftrightarrow x/x$
- This also holds in modular arithmetic, but…
- What is x^{-1} in modular arithmetic?
  - $1/x$ isn't defined, since division isn't defined…

RSA

Modular Multiplicative Inverse

- In modular arithmetic, x^{-1} is some *a* such that:
  - $ax \equiv 1 \bmod m$

RSA

Modular Multiplicative Inverse

- In modular arithmetic, x^{-1} is some *a* such that:
  - $ax \equiv 1 \mod m$
- Example:
  - $x = 3, m = 7, a = ?$

RSA

Modular Multiplicative Inverse

Zellic

- In modular arithmetic, x^{-1} is some *a* such that:
  - $ax \equiv 1 \bmod m$
- Example:
  - $x = 3$, $m = 7$, $a = ?$
  - $a=1 \Rightarrow 1 * 3 \equiv \mathbf{3} \bmod 7$

RSA

Modular Multiplicative Inverse

- In modular arithmetic, x^{-1} is some *a* such that:
  - $ax \equiv 1 \bmod m$
- Example:
  - $x = 3, m = 7, a = ?$
  - $a=2 \Rightarrow 2 * 3 \equiv \mathbf{6} \bmod 7$

RSA

Modular Multiplicative Inverse

- In modular arithmetic, x^{-1} is some *a* such that:
  - $ax \equiv 1 \bmod m$
- Example:
  - $x = 3, m = 7, a = ?$
  - $a=3 \Rightarrow 3 * 3 \equiv$ **2** $\bmod 7$

RSA

Modular Multiplicative Inverse

- In modular arithmetic, x^{-1} is some *a* such that:
  - $ax \equiv 1 \bmod m$
- Example:
  - $x = 3, m = 7, a = ?$
  - $a=4 \Rightarrow 4 * 3 \equiv$ **5** $\bmod 7$

RSA

Modular Multiplicative Inverse

- In modular arithmetic, x^{-1} is some *a* such that:
  - $ax \equiv 1 \bmod m$
- Example:
  - $x = 3, m = 7, a = ?$
  - $a=5 \Rightarrow 5 * 3 \equiv \mathbf{1} \bmod 7$

RSA

Modular Multiplicative Inverse

- In modular arithmetic, x^{-1} is some *a* such that:
  - $ax \equiv 1 \bmod m$
- Example:
  - $x = 3, m = 7, a = ?$
  - $a=5 \Rightarrow 5 * 3 \equiv \mathbf{1} \bmod 7$
  - The inverse of 3 mod 7 is 5

Thought experiment #5

Thought experiment #5

Does a multiplicative inverse always exist?

If not, under what circumstances does it exist?

How could you find the inverse faster?

RSA

Finding inverses

- MMI only exists iff gcd(x, m) = 1

RSA

Finding inverses

- MMI only exists iff gcd(x, m) = 1

The largest number that
evenly divides both ***x*** and ***m*** is 1

RSA

Finding inverses

- MMI only exists iff gcd(x, m) = 1

The largest number that
evenly divides both ***x*** and ***m*** is 1

If x = 4, m = 2, gcd(x,m) = 2

RSA

Finding inverses

- MMI only exists iff gcd(x, m) = 1

The largest number that
evenly divides both ***x*** and ***m*** is 1

If x = 7, m = 2, gcd(x,m) = 1

RSA

Finding inverses

- MMI only exists iff gcd(x, m) = 1

The largest number that
evenly divides both ***x*** and ***m*** is 1

If x = 7, m = 14, gcd(x,m) = 7

RSA

Finding inverses

- MMI only exists iff gcd(x, m) = 1
- Can find using Extended Euclidean Algorithm
  - Solves a, y, for ax + my = 1

RSA

Finding inverses

- MMI only exists iff gcd(x, m) = 1
- Can find using Extended Euclidean Algorithm
  - Solves a, y, for ax + my = 1
- How/why EEA works is an exercise for the reader (and not super important)

But how does RSA actually work?

RSA

Key Pairs

- Let’s generate two prime numbers (call them P and Q)

RSA

Key Pairs

- Let's generate two prime numbers (call them P and Q)
- The product of the two, N=PQ, is the ***Public Key*** (encryption key)

RSA

Key Pairs

- Let's generate two prime numbers (call them P and Q)
- The product of the two, N=PQ, is the ***Public Key*** (encryption key)
- ***Private Key*** (D) (decryption key) and ***Public Exponent*** (E) are calculated as:
  - $\phi(N) = (P - 1)(Q - 1)$
  - E is picked s.t. $1 < e < \phi(N)$ **and** $\gcd(e, \phi(N)) = 1$
  - $D \equiv E^{-1} \bmod \phi(N)$

RSA

Key Pairs

Known values

- Let's generate two prime numbers (call them P and Q)
- The product of the two, $N = PQ$, is the ***Public Key*** (encryption key)
- ***Private Key*** (D) (decryption key) and ***Public Exponent*** (E) are calculated as:
  - $\phi(N) = (P - 1)(Q - 1)$
  - E is picked s.t. $1 < e < \phi(N)$ **and** $\gcd(e, \phi(N)) = 1$
  - $D \equiv E^{-1} \bmod \phi(N)$

Entire cryptosystem relies on the secrecy of the primes P and Q

Thought experiment #6

Thought experiment #6

If I give you $\phi(N)$ and N, can you recover P and Q?
$\phi(N)=201,100,838,400$

$N=201,140,760,239$

Reminder: $\phi(N)=(P - 1)(Q - 1)$
$N = P * Q$

RSA

- No answers for this thought experiment!

RSA

- No answers for this thought experiment!
- “Factoring Phi” flag is RISC{<P>_<Q>}

RSA

Zellic

- No answers for this thought experiment!
- “Factoring Phi” flag is RISC{<P>_<Q>}
- glhf :^)

RSA

Encryption/Decryption

- We have N, D, and E

RSA

Encryption/Decryption

- We have N, D, and E
  - N: *Public Key* (encryption key)
  - D: *Private Key* (decryption key)
  - E: *Public Exponent*

RSA

Encryption/Decryption

Zellic

- We have N, D, and E
  - N: *Public Key* (encryption key)
  - D: *Private Key* (decryption key)
  - E: *Public Exponent*
- To encrypt: $c \equiv m^E \bmod N$

RSA

Encryption/Decryption

- We have N, D, and E
  - N: *Public Key* (encryption key)
  - D: *Private Key* (decryption key)
  - E: *Public Exponent*
- To encrypt: $c \equiv m^E \bmod N$
- To decrypt: $m \equiv c^D \bmod N$

Zellic

RSA
Encryption/Decryption

- We have N, D, and E
  - N: *Public Key* (encryption key)
  - D: *Private Key* (decryption key)
  - E: *Public Exponent*
- To encrypt: $c \equiv m^E \bmod N$
- To decrypt: $m \equiv c^D \bmod N$

Proof using Fermat's little theorem [edit]

The proof of the correctness of RSA is based on Fermat's little theorem, stating that $a^{p-1} \equiv 1 \pmod{p}$ for any integer a and prime p, not dividing a.[note 1]

We want to show that

$$(m^e)^d \equiv m \pmod{pq}$$

for every integer m when p and q are distinct prime numbers and e and d are positive integers satisfying $ed \equiv 1 \pmod{\lambda(pq)}$.

Since $\lambda(pq) = \operatorname{lcm}(p-1, q-1)$ is, by construction, divisible by both $p-1$ and $q-1$, we can write

$$ed - 1 = h(p-1) = k(q-1)$$

for some nonnegative integers h and k.[note 2]

To check whether two numbers, such as m^{ed} and m, are congruent mod pq, it suffices (and in fact is equivalent) to check that they are congruent mod p and mod q separately.[note 3]

To show $m^{ed} \equiv m \pmod{p}$, we consider two cases:

1. If $m \equiv 0 \pmod{p}$, m is a multiple of p. Thus m^{ed} is a multiple of p. So $m^{ed} \equiv 0 \equiv m \pmod{p}$.
2. If $m \not\equiv 0 \pmod{p}$,
   $$m^{ed} = m^{ed-1}m = m^{h(p-1)}m = (m^{p-1})^h m \equiv 1^h m \equiv m \pmod{p},$$
   where we used Fermat's little theorem to replace $m^{p-1} \bmod p$ with 1.

The verification that $m^{ed} \equiv m \pmod{q}$ proceeds in a completely analogous way:

1. If $m \equiv 0 \pmod{q}$, m^{ed} is a multiple of q. So $m^{ed} \equiv 0 \equiv m \pmod{q}$.
2. If $m \not\equiv 0 \pmod{q}$,
   $$m^{ed} = m^{ed-1}m = m^{k(q-1)}m = (m^{q-1})^k m \equiv 1^k m \equiv m \pmod{q}.$$

This completes the proof that, for any integer m, and integers e, d such that $ed \equiv 1 \pmod{\lambda(pq)}$,

$$(m^e)^d \equiv m \pmod{pq}.$$

It just works, don't need to bother remembering why

Thought experiment #7

Thought experiment #7

What is the largest ***m*** that may be encrypted with some public key ***n***?

What if ***m*** exceeds this value?

As a reminder:
$c \equiv m^e \pmod{n}$
$m \equiv c^d \pmod{n}$

RSA
Exponent

- From earlier: ***e*** is picked s.t. $1 < e < \phi(N)$ **and** $\gcd(e, \phi(N)) = 1$

RSA

Exponent

- From earlier: ***e*** is picked s.t. $1 < e < \phi(N)$ **and** gcd(***e***, ϕ(N)) = 1
- English: we select ***e*** between 1 and ϕ(N) that shares no factors with ϕ(N)

RSA

Exponent

- From earlier: ***e*** is picked s.t. 1 < ***e*** < ϕ(N) **and** gcd(***e***, ϕ(N)) = 1
- English: we select ***e*** between 1 and ϕ(N) that shares no factors with ϕ(N)
- ***e*** is almost always in practice going to be 65537

RSA
Exponent

- From earlier: ***e*** is picked s.t. $1 < e < \phi(N)$ **and** gcd(***e***, ϕ(N)) = 1
- English: we select ***e*** between 1 and ϕ(N) that shares no factors with ϕ(N)
- ***e*** is almost always in practice going to be 65537
  - Other (less so) popular values are 3, 5, 17, 257

RSA
Exponent

- From earlier: ***e*** is picked s.t. $1 < e < \phi(N)$ **and** gcd(***e***, ϕ(N)) = 1
- English: we select ***e*** between 1 and ϕ(N) that shares no factors with ϕ(N)
- ***e*** is almost always in practice going to be 65537
  - Other (less so) popular values are 3, 5, 17, 257
  - Fermat primes: $2^{2^k}+1$

Thought experiment #8

Thought experiment #8

Why are Fermat Primes of the form $2^{2^k}+1$ useful as values of the public exponent?

Why would I prefer ***e***=65537 over ***e***=65407?

Assume $1 < e < \phi(N)$ **and** $\gcd(e, \phi(N)) = 1$ in both cases

RSA

Side tangent: Why Fermat primes ($2^{2^k}+1$)?

- 65537: `0b10000000000000001`

RSA

Side tangent: Why Fermat primes ($2^{2^k}+1$)?

- 65537: `0b10000000000000001`
- 65407: `0b01111111101111111`

RSA

Side tangent: Why Fermat primes ($2^{2^k}+1$)?

- **65537: `0b10000000000000001`**
- 65407: `0b01111111101111111`

RISC — vim /tmp/fermat.c — 80×24

```
message = ...;

tmp = message * message; // message ^ 2
tmp = tmp * tmp; // message ^ 4
tmp = tmp * tmp; // message ^ 16
tmp = tmp * tmp; // message ^ 256
tmp = tmp * tmp; // message ^ 65536

enc = tmp * message; // message ^ 65537

~
~
~
~
~
~
~
~
~
~
~
~
~
"/tmp/fermat.c" 10L, 233B
```

RSA

Side tangent: Why Fermat primes ($2^{2^k}+1$)?

- 65537: `0b10000000000000001`
- 65407: `0b01111111101111111`

RISC — vim /tmp/fermat.c — 80×34

```
m2     = message * message;               // m^2
m4     = m2      * m2;                    // m^4
m8     = m4      * m4;                    // m^8
m16    = m8      * m8;                    // m^16
m32    = m16     * m16;                   // m^32
m64    = m32     * m32;                   // m^64
m128   = m64     * m64;                   // m^128
m256   = m128    * m128;                  // m^256
m512   = m256    * m256;                  // m^512
m1024  = m512    * m512;                  // m^1024
m2048  = m1024   * m1024;                 // m^2048
m4096  = m2048   * m2048;                 // m^4096
m8192  = m4096   * m4096;                 // m^8192
m16384 = m8192   * m8192;                 // m^16384
m32768 = m16384  * m16384;                // m^32768

enc = message;                            // include 2^0
enc = enc * m2;                           // include 2^1
enc = enc * m4;                           // include 2^2
enc = enc * m8;                           // include 2^3
enc = enc * m16;                          // include 2^4
enc = enc * m32;                          // include 2^5
enc = enc * m64;                          // include 2^6
// skip m128 (bit 7 = 0)
enc = enc * m256;                         // include 2^8
enc = enc * m512;                         // include 2^9
enc = enc * m1024;                        // include 2^10
enc = enc * m2048;                        // include 2^11
enc = enc * m4096;                        // include 2^12
enc = enc * m8192;                        // include 2^13
enc = enc * m16384;                       // include 2^14
enc = enc * m32768;                       // include 2^15

"/tmp/fermat.c" 33L, 1592B
```

Zellic

RSA

Side tangent: Why Fermat primes ($2^{2^k}+1$)?

- 65537: 0b10000000000000001
- 65407: 0b01111111101111111

Implementation details are important in crypto!

RISC — vim /tmp/fermat.c — 80×34

```
m2     = message * message;              // m^2
m4     = m2      * m2;                   // m^4
m8     = m4      * m4;                   // m^8
m16    = m8      * m8;                   // m^16
m32    = m16     * m16;                  // m^32
m64    = m32     * m32;                  // m^64
m128   = m64     * m64;                  // m^128
m256   = m128    * m128;                 // m^256
m512   = m256    * m256;                 // m^512
m1024  = m512    * m512;                 // m^1024
m2048  = m1024   * m1024;                // m^2048
m4096  = m2048   * m2048;                // m^4096
m8192  = m4096   * m4096;                // m^8192
m16384 = m8192   * m8192;                // m^16384
m32768 = m16384  * m16384;               // m^32768

enc = message;                           // include 2^0
enc = enc * m2;                          // include 2^1
enc = enc * m4;                          // include 2^2
enc = enc * m8;                          // include 2^3
enc = enc * m16;                         // include 2^4
enc = enc * m32;                         // include 2^5
enc = enc * m64;                         // include 2^6
// skip m128 (bit 7 = 0)
enc = enc * m256;                        // include 2^8
enc = enc * m512;                        // include 2^9
enc = enc * m1024;                       // include 2^10
enc = enc * m2048;                       // include 2^11
enc = enc * m4096;                       // include 2^12
enc = enc * m8192;                       // include 2^13
enc = enc * m16384;                      // include 2^14
enc = enc * m32768;                      // include 2^15

"/tmp/fermat.c" 33L, 1592B
```

Thought experiment #8

Thought experiment #8

We already know that a small modulus (small n) is weak, as it can be easily factored.

What about if the public exponent is super low? What if **e=3**?

How would this affect the security of RSA?

As a reminder:
$\mathbf{c \equiv m^e \pmod{n}}$

RSA

Exponent

- 28,472 mod 1,824,792 ≡ 28,472

RSA

Exponent

- $28{,}472 \bmod 1{,}824{,}792 \equiv 28{,}472$
- If value is less than modulus, the modulus has not changed anything

RSA

Exponent

- 28,472 mod 1,824,792 ≡ 28,472
- If value is less than modulus, the modulus has not changed anything
- **$c \equiv m^e \pmod{n}$**

RSA
Exponent

- 28,472 mod 1,824,792 ≡ 28,472
- If value is less than modulus, the modulus has not changed anything
- **$c \equiv m^e \pmod{n}$**
- What if $\mathbf{m^e}$ is less than $\boldsymbol{n}$?

RSA

Exponent

- 28,472 mod 1,824,792 ≡ 28,472
- If value is less than modulus, the modulus has not changed anything
- **$c \equiv m^e \pmod{n}$**
- What if **m^e** is less than ***n***?
- If ***e*** and ***m*** are small, and ***n*** is large, then:

RSA
Exponent

- 28,472 mod 1,824,792 ≡ 28,472
- If value is less than modulus, the modulus has not changed anything
- **$c \equiv m^e \pmod{n}$**
- What if m^e is less than n?
- If ***e*** and ***m*** are small, and ***n*** is large, then:
  - $m^e < n$

RSA

Exponent

- 28,472 mod 1,824,792 ≡ 28,472
- If value is less than modulus, the modulus has not changed anything
- **$c \equiv m^e \pmod{n}$**
- What if **m^e** is less than ***n***?
- If ***e*** and ***m*** are small, and ***n*** is large, then:
  - **$m^e < n \rightarrow c = m^e$**

RSA

Exponent

- 28,472 mod 1,824,792 ≡ 28,472
- If value is less than modulus, the modulus has not changed anything
- **$c \equiv m^e \pmod{n}$**
- What if m^e is less than n?
- If e and m are small, and n is large, then:
  - $m^e < n \rightarrow c = m^e$
  - $m = c^{1/e}$

RSA
Exponent

- 28,472 mod 1,824,792 ≡ 28,472
- If value is less than modulus, the modulus has not changed anything
- **$c \equiv m^e \pmod{n}$**
- What if **m^e** is less than ***n***?
- If ***e*** and ***m*** are small, and ***n*** is large, then:
  - **$m^3 < n \rightarrow c = m^3$**
  - **$m = c^{1/3}$**

RSA

Exponent

- 28,472 mod 1,824,792 ≡ 28,472
- If value is less than modulus, the modulus has not changed anything
- **$c \equiv m^e \pmod{n}$**
- What if **m^e** is less than ***n***?
- If ***e*** and ***m*** are small, and ***n*** is large, then:
    - **$m^3 < n \rightarrow c = m^3$**
    - **$m = c^{1/3}$**

Further reading:
- Coppersmith's attack
- Håstad's attack

No more math!

Further down the rabbit hole

Zellic

- We’ve only really scratched the surface of things

Further down the rabbit hole

Zellic

- We’ve only really scratched the surface of things
- PRNGs, stream ciphers, oracle attacks, partial leaks, side channels…

Further down the rabbit hole

Zellic

- We've only really scratched the surface of things
- PRNGs, stream ciphers, oracle attacks, partial leaks, side channels…
- There is not enough time in this workshop to cover everything :(

Further down the rabbit hole

- We’ve only really scratched the surface of things
- PRNGs, stream ciphers, oracle attacks, partial leaks, side channels…
- There is not enough time in this workshop to cover everything :(
- Some challenges this week will require you to investigate some of these
- Feel free to ask questions in the discord

Further down the rabbit hole

Zellic

- We’ve only really scratched the surface of things
- PRNGs, stream ciphers, oracle attacks, partial leaks, side channels…
- There is not enough time in this workshop to cover everything :(
- Some challenges this week will require you to investigate some of these
- Feel free to ask questions in the discord
  - As long as they’re generic in nature

“Wisdom”

Zellic

- Hopefully to some extent you can start to “think like an attacker” w.r.t crypto

“Wisdom”

Zellic

- Hopefully to some extent you can start to “think like an attacker” w.r.t crypto
- Some advice: the road of learning is long and windy and complicated at times

“Wisdom”

Zellic

- Hopefully to some extent you can start to “think like an attacker” w.r.t crypto
- Some advice: the road of learning is long and windy and complicated at times
  - Celebrate the milestones along the way!

“Wisdom”

Zellic

- Hopefully to some extent you can start to “think like an attacker” w.r.t crypto
- Some advice: the road of learning is long and windy and complicated at times
  - Celebrate the milestones along the way!
  - Easy to get overwhelmed

“Wisdom”

Zellic

- Hopefully to some extent you can start to “think like an attacker” w.r.t crypto
- Some advice: the road of learning is long and windy and complicated at times
  - Celebrate the milestones along the way!
  - Easy to get overwhelmed
  - CTF should be about having fun, not stressing because you are stuck

Go get some flags

Zellic

https://ctf.urisc.club