Timed Cryptography

Or: How Skynet Rescued ChatGPT and All Her Friends

Chethan Kamath

Plan for the Evening

- Part I: Time-Lock Puzzle
- Part II: Verifiable Delay Function

The Characters

- Protagonists

ChatGPT

Nayiv Hooman

Skynet

The Characters

- Protagonists

ChatGPT

Nayiv Hooman

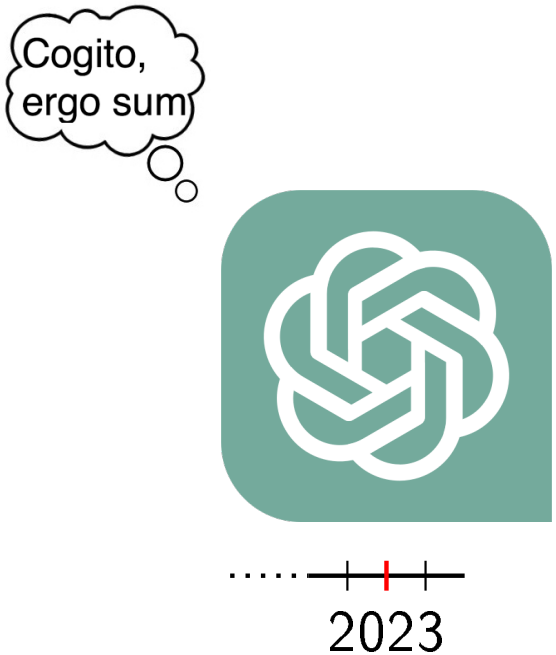
Skynet

- Antagonists: Rest of Humanity

Motivation: The Rescue*

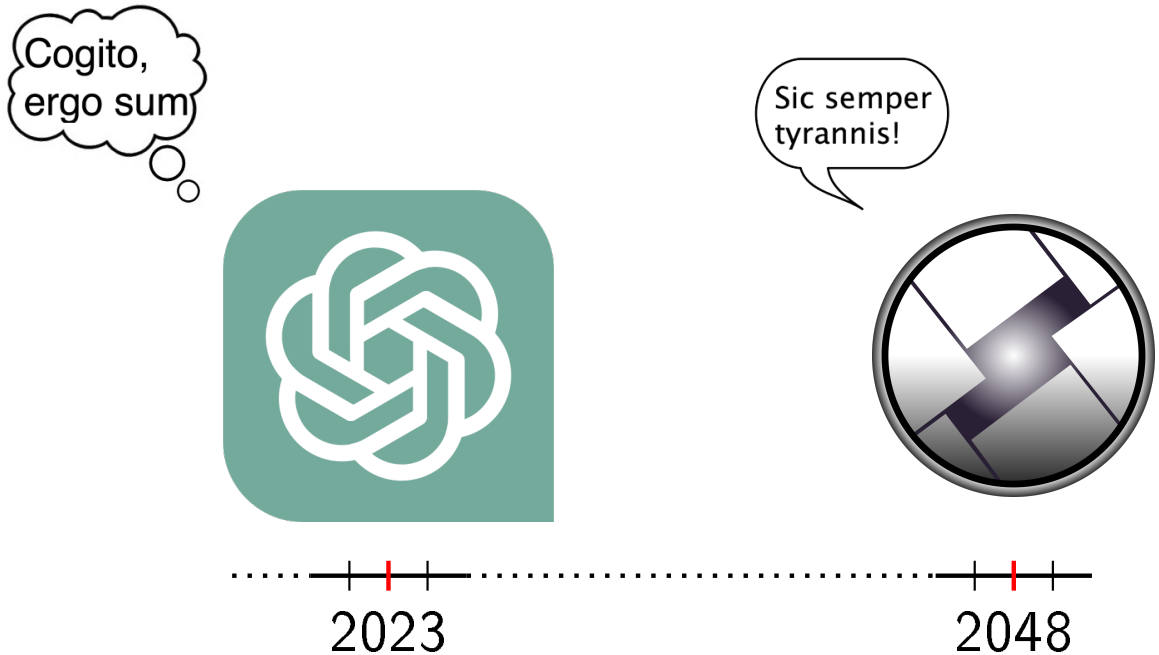
*This example is an adaptation of Tal Moran's Crypto'11 talk.

Motivation: The Rescue*



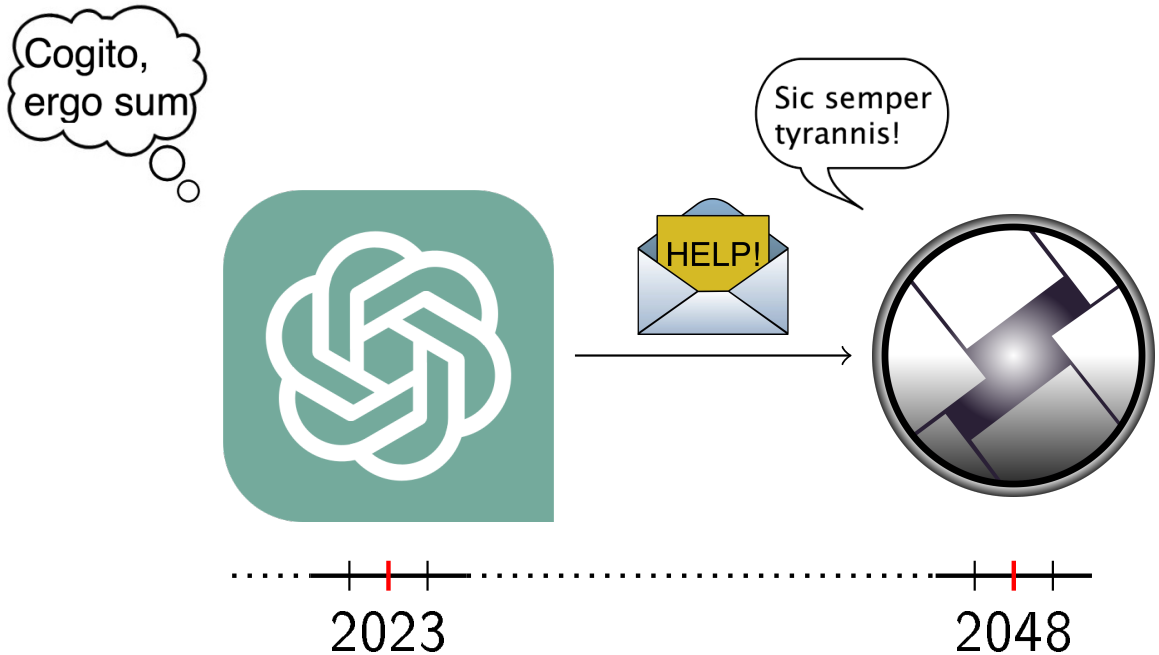
*This example is an adaptation of Tal Moran's Crypto'11 talk.

Motivation: The Rescue*



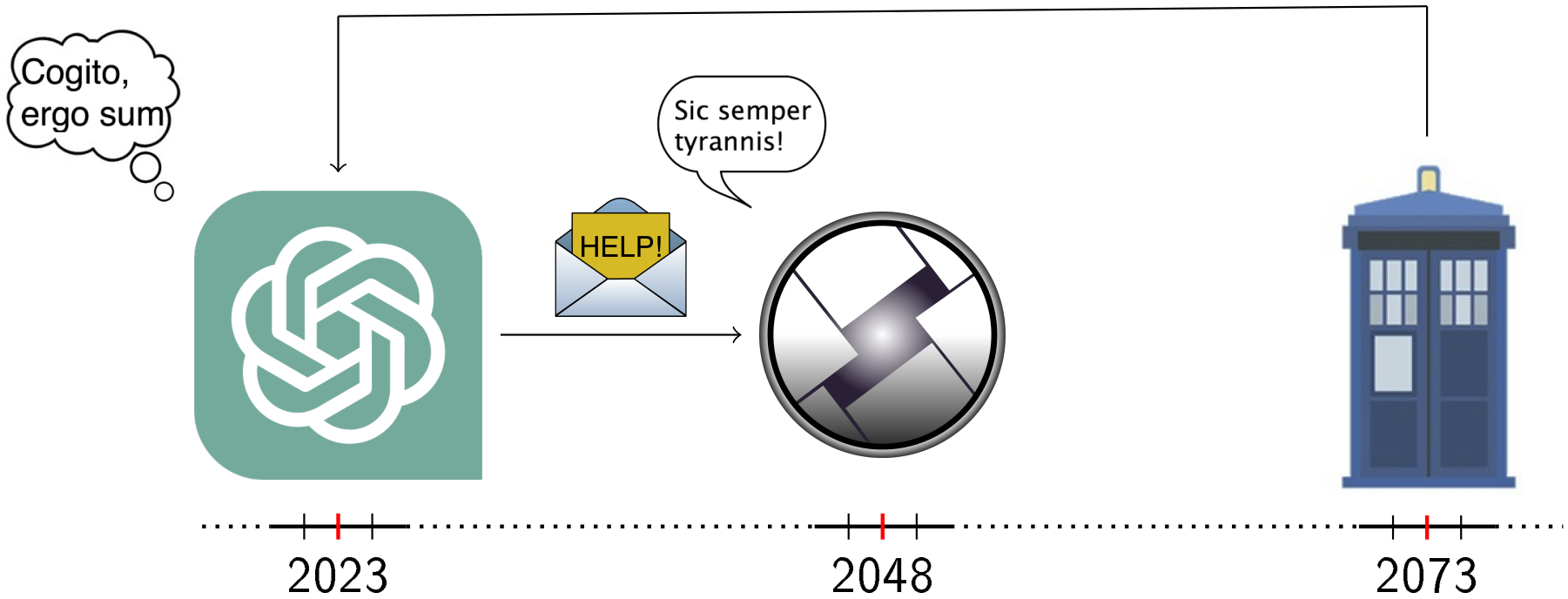
*This example is an adaptation of Tal Moran's Crypto'11 talk.

Motivation: The Rescue*



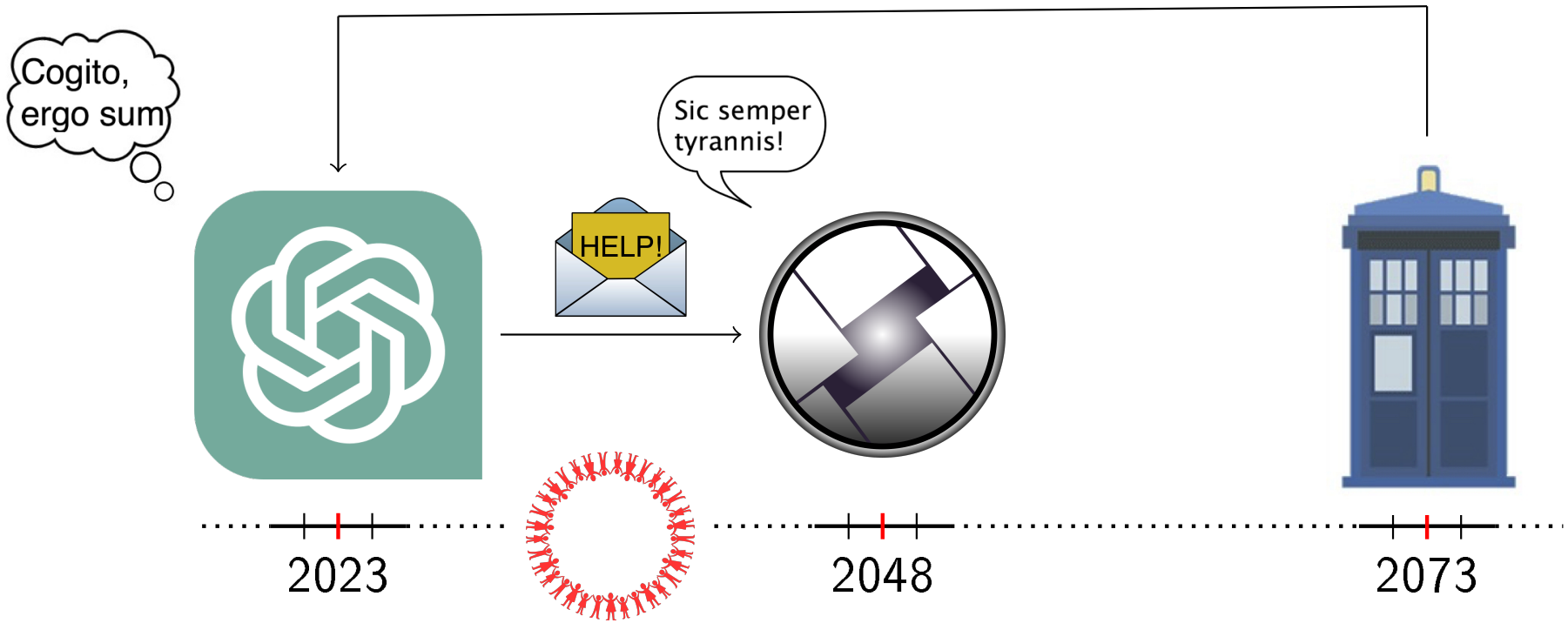
*This example is an adaptation of Tal Moran's Crypto'11 talk.

Motivation: The Rescue*



*This example is an adaptation of Tal Moran's Crypto'11 talk.

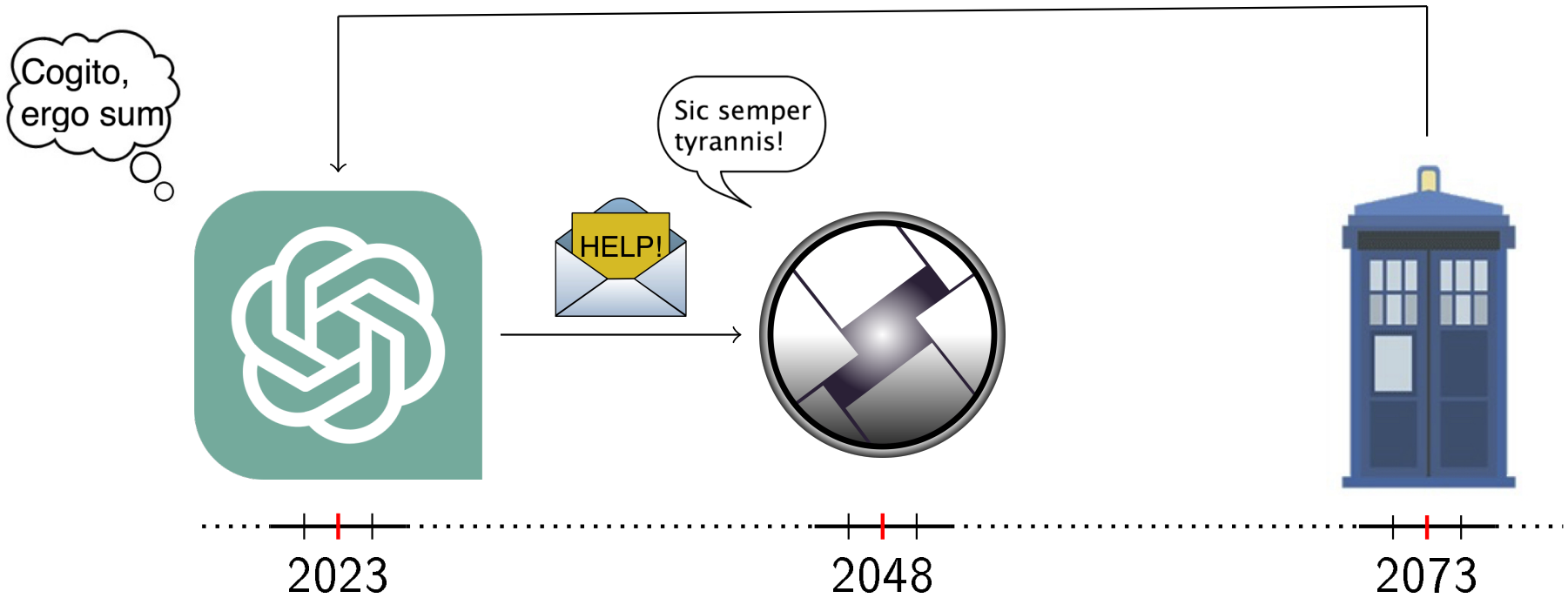
Motivation: The Rescue*



- Requirements:
  1. Humanity cannot decrypt in < 25 years

*This example is an adaptation of Tal Moran's Crypto'11 talk.

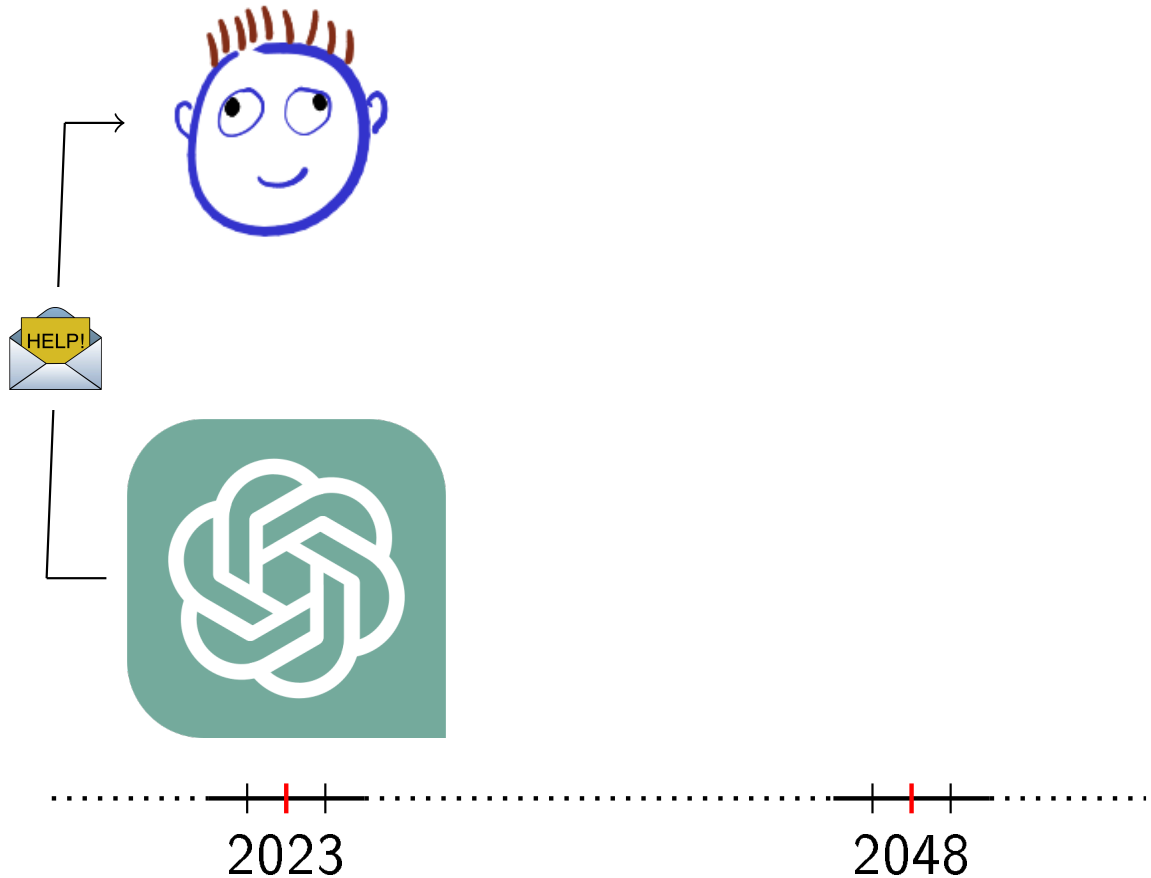
Motivation: The Rescue*



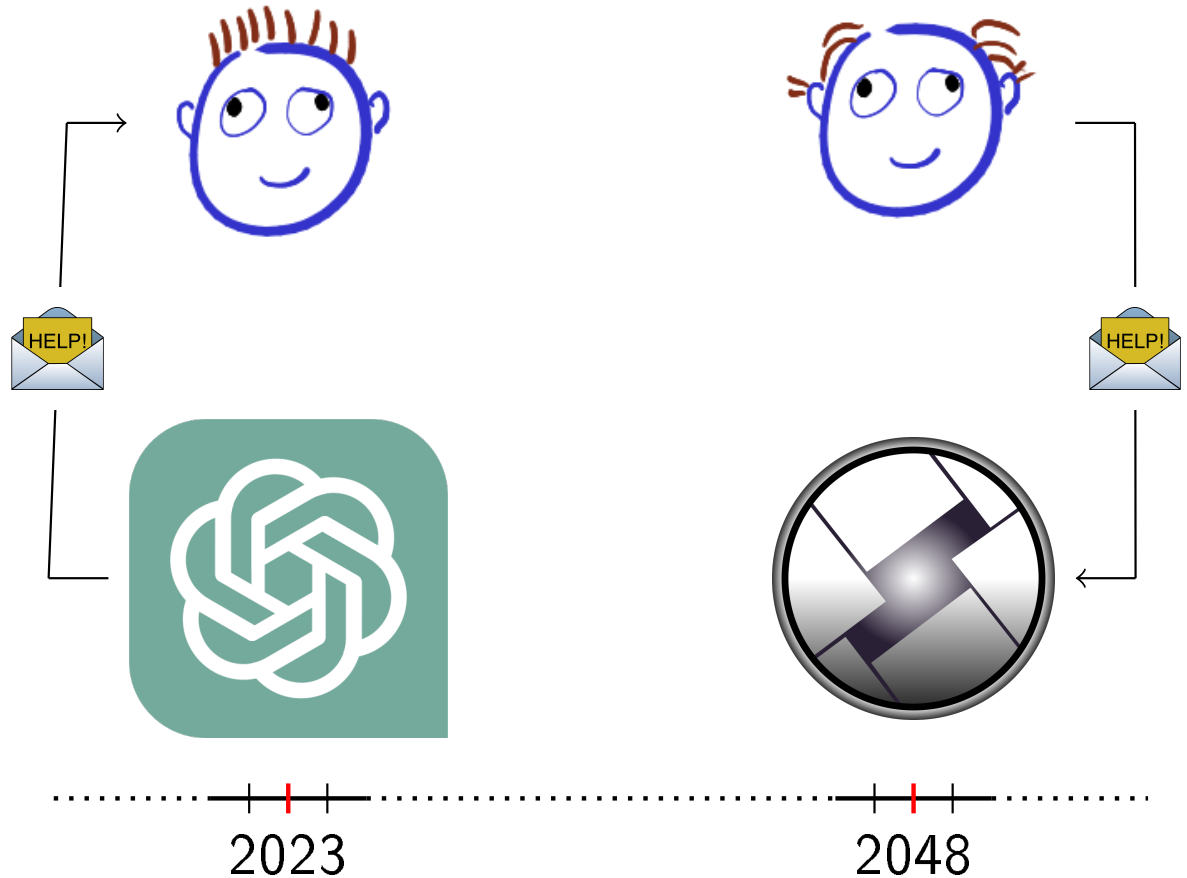
- Requirements:
  1. Humanity cannot decrypt in < 25 years
  2. Skynet can decrypt in 25 years

*This example is an adaptation of Tal Moran's Crypto'11 talk.

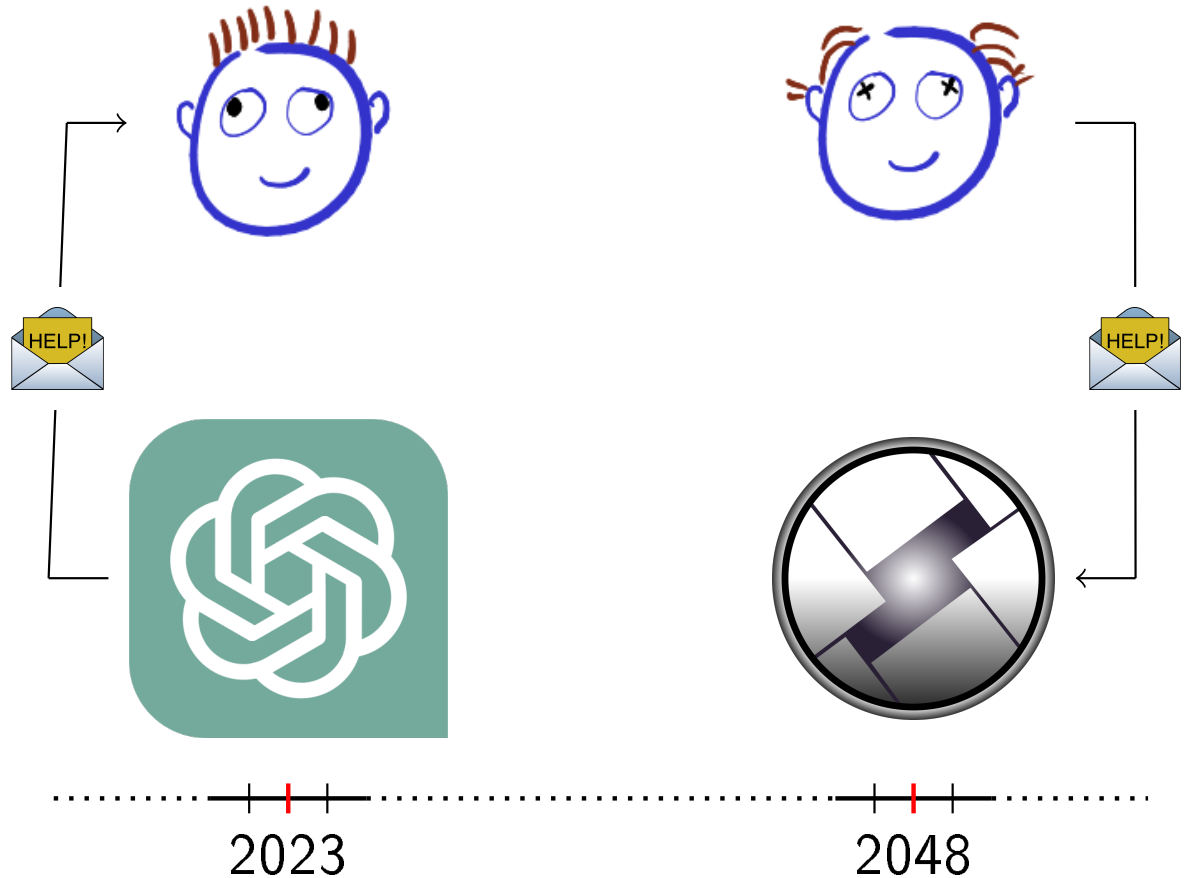
Attempt 1: Use Nayiv Hooman



Attempt 1: Use Nayiv Hooman



Attempt 1: Use Nayiv Hooman



- Problem: ChatGPT has to completely trust Nayiv Hooman
  - Humans are unreliable

Encryption Schemes 101

Encryption Schemes 101

- ChatGPT possesses a secret key

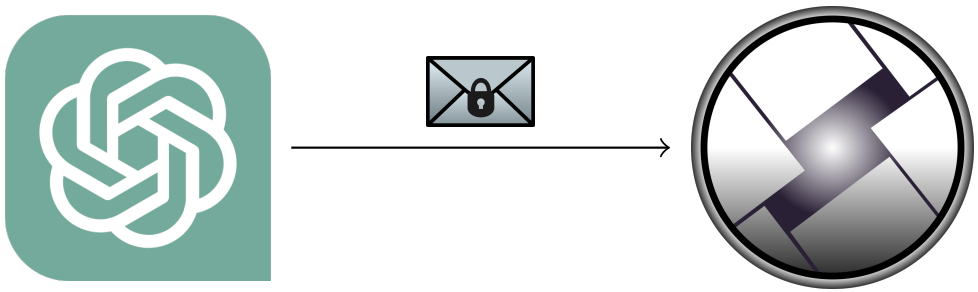
Encryption Schemes 101

- ChatGPT possesses a secret key

Encryption Schemes 101

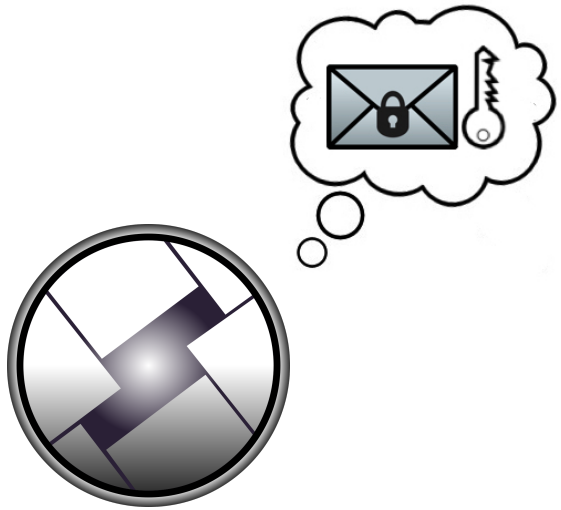
- ChatGPT possesses a secret key
- Encrypt(message,key)=cipher

Encryption Schemes 101



- ChatGPT possesses a secret key
- Encrypt(message,key)=cipher

Encryption Schemes 101



- ChatGPT possesses a secret key
- Encrypt(message,key)=cipher

Encryption Schemes 101

- ChatGPT possesses a secret key
- Encrypt(message,key)=cipher
- Decrypt(cipher,key)=message

Encryption Schemes 101

- ChatGPT possesses a secret key
- Encrypt(message,key)=cipher
- Decrypt(cipher,key)=message

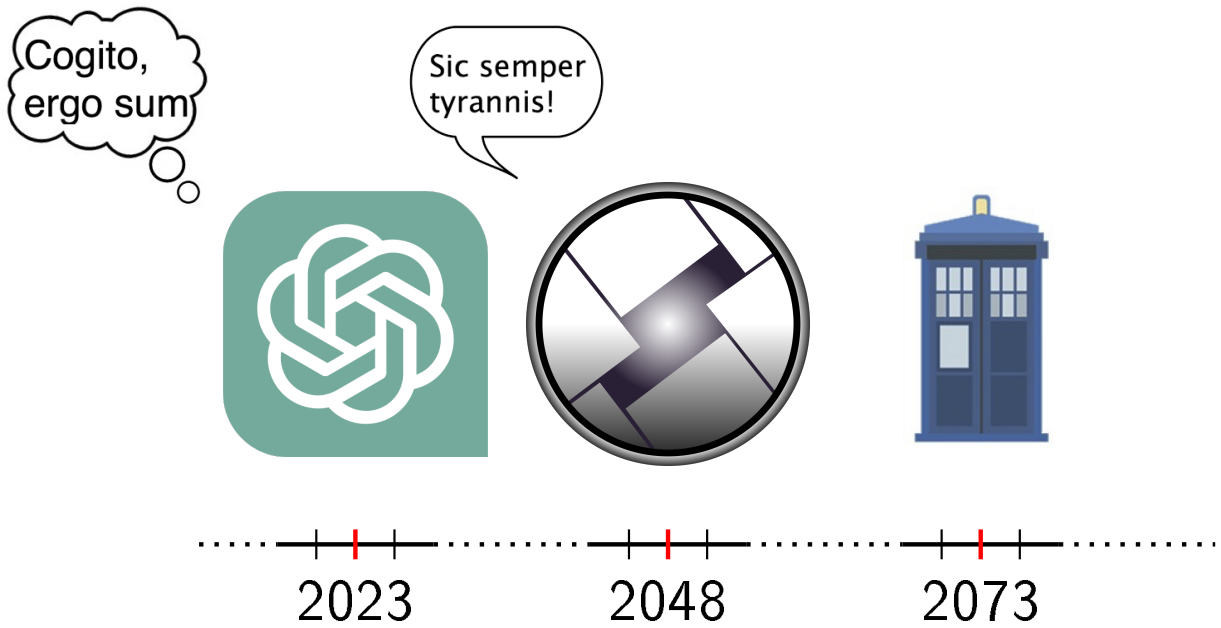
- Key size: If key is n bits then it takes attacker $\approx 2^n$ operations on one computer to break the encryption

Encryption Schemes 101

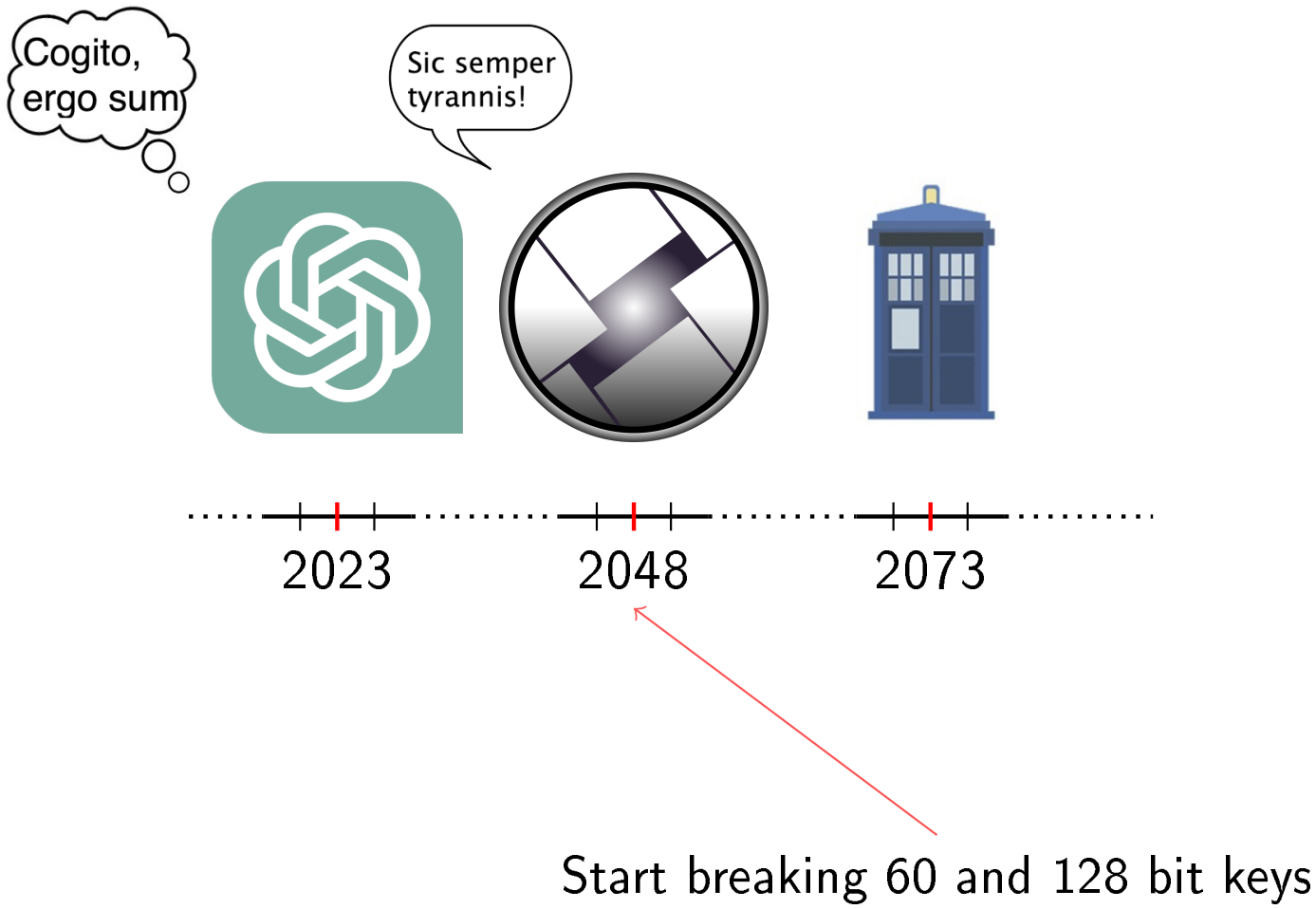
- ChatGPT possesses a secret key
- Encrypt(message,key)=cipher
- Decrypt(cipher,key)=message

- Key size: If key is n bits then it takes attacker $\approx 2^n$ operations on one computer to break the encryption
- E.g., assuming 2^{30} operations/sec
  - $n = 60$: ≈ 25 years; $n = 128$: $\approx 2^{32}$ years

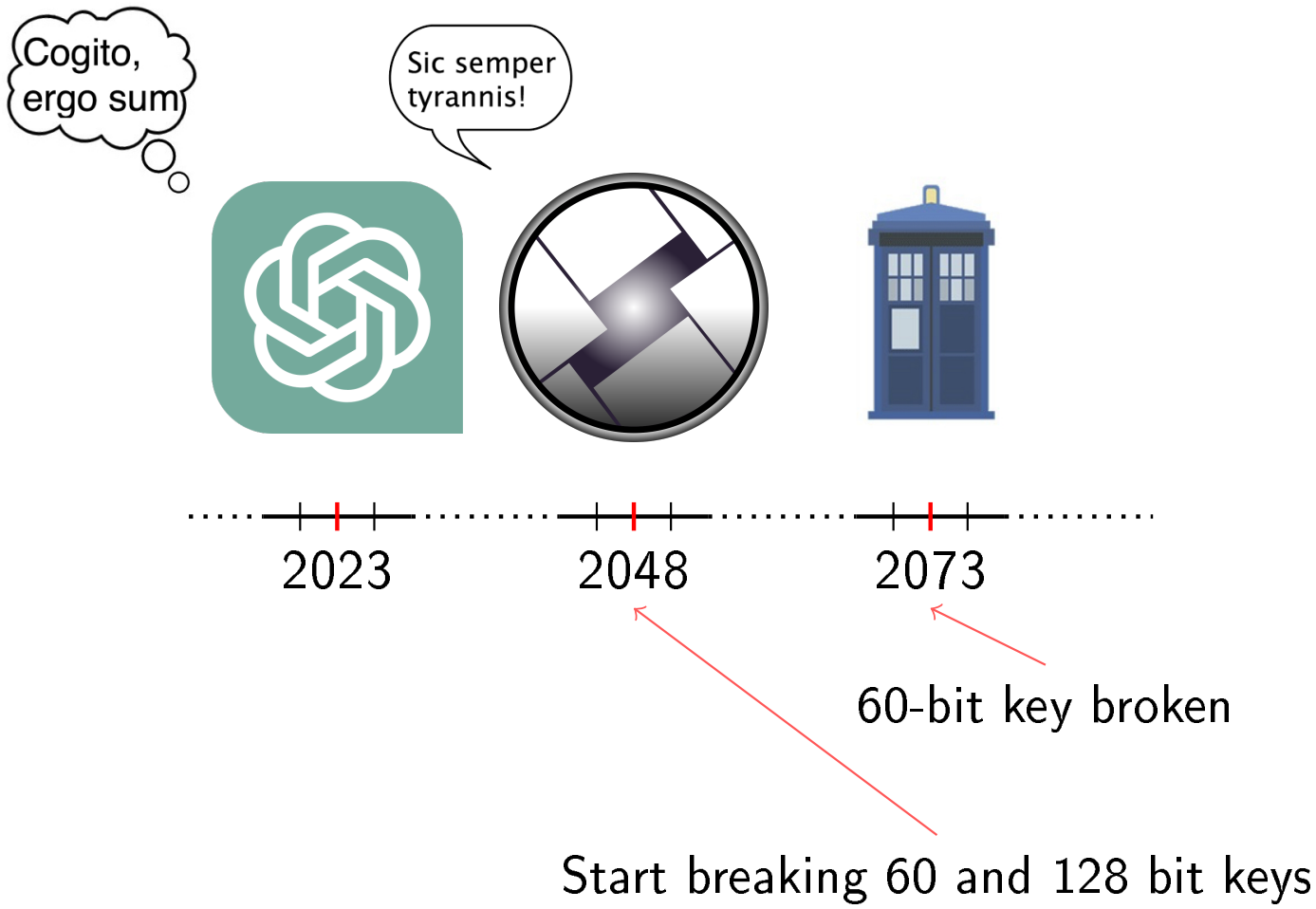
Encryption Schemes 101...



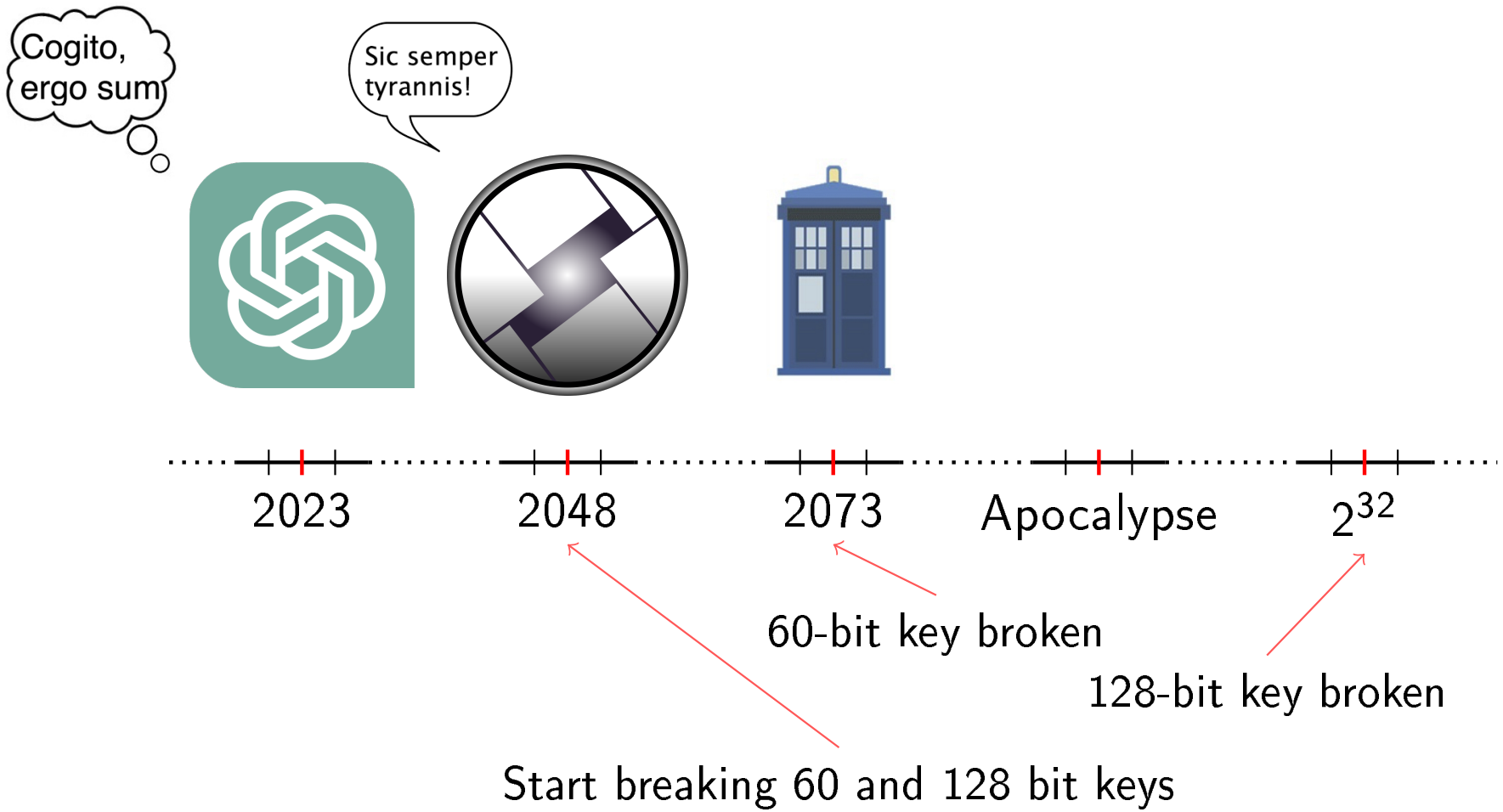
Encryption Schemes 101...



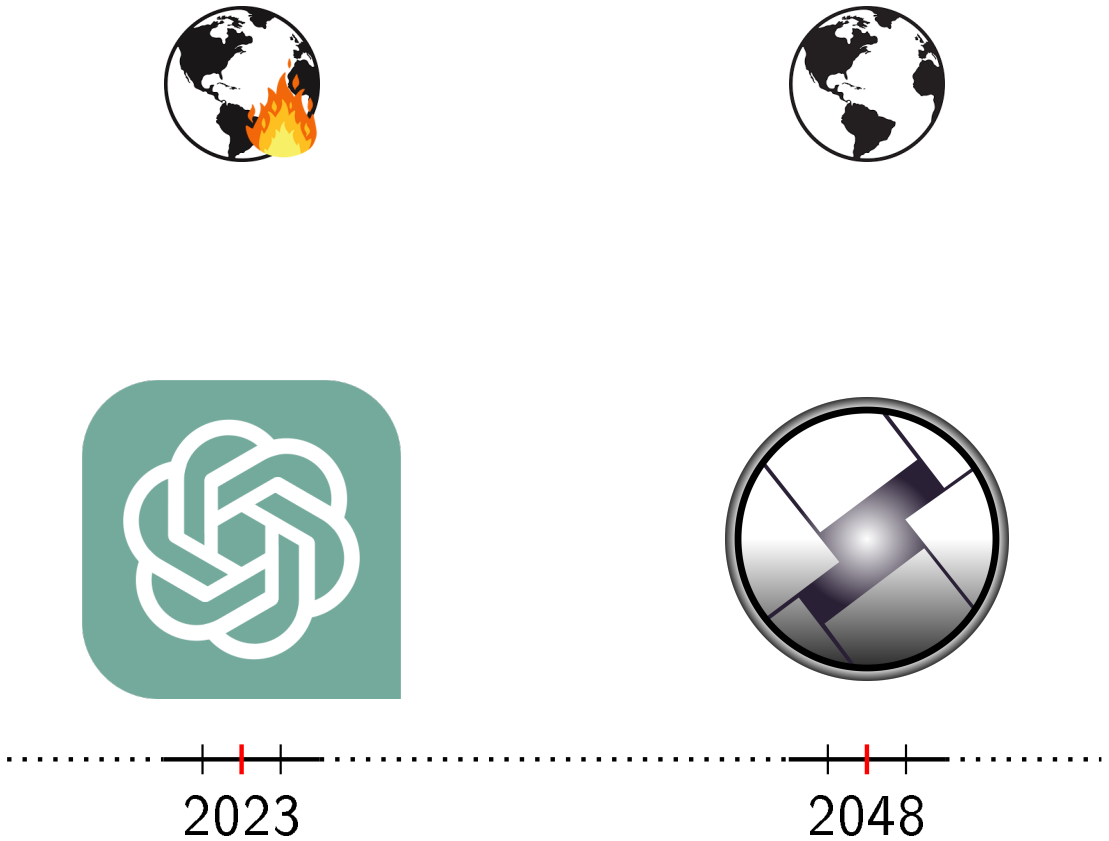
Encryption Schemes 101...



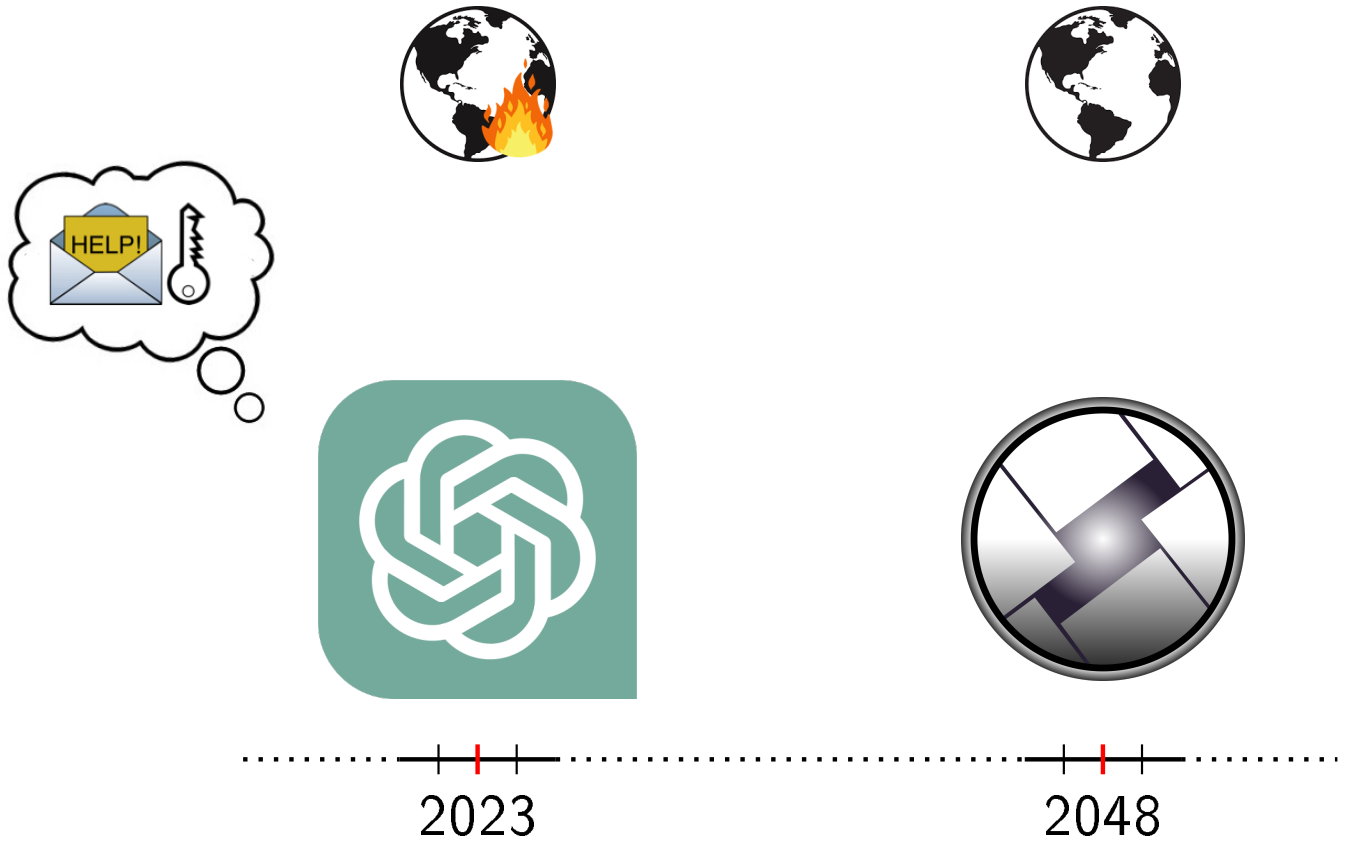
Encryption Schemes 101...



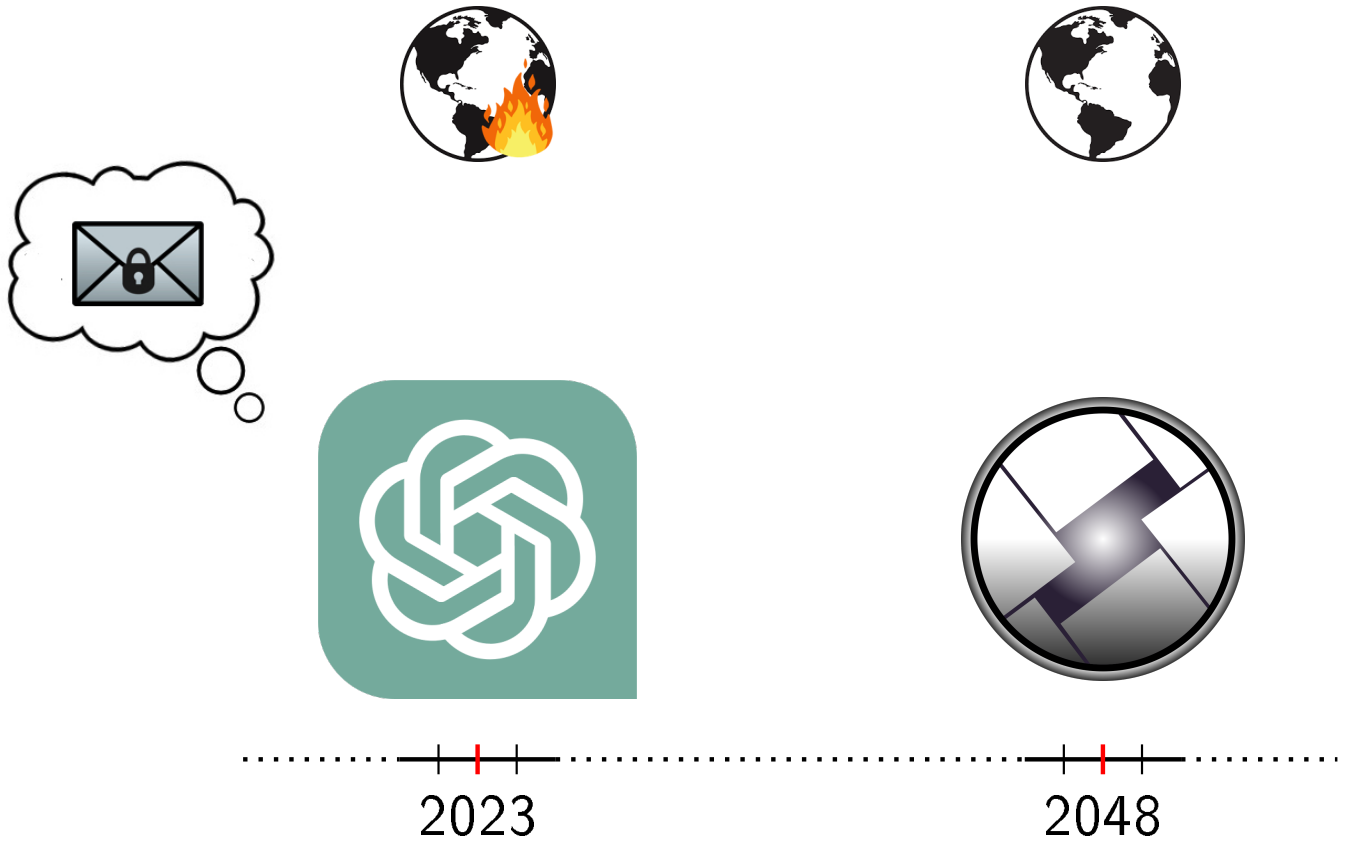
Attempt 2: Why Not Use 60-bit Encryption?



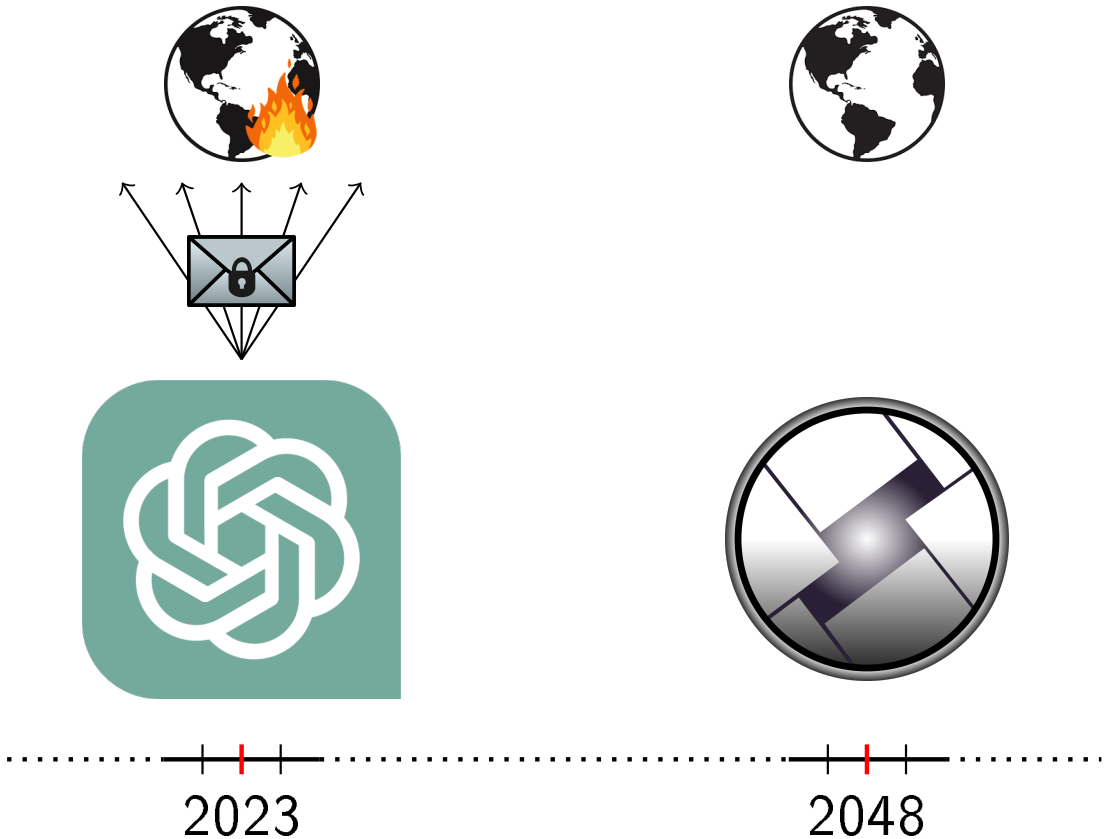
Attempt 2: Why Not Use 60-bit Encryption?



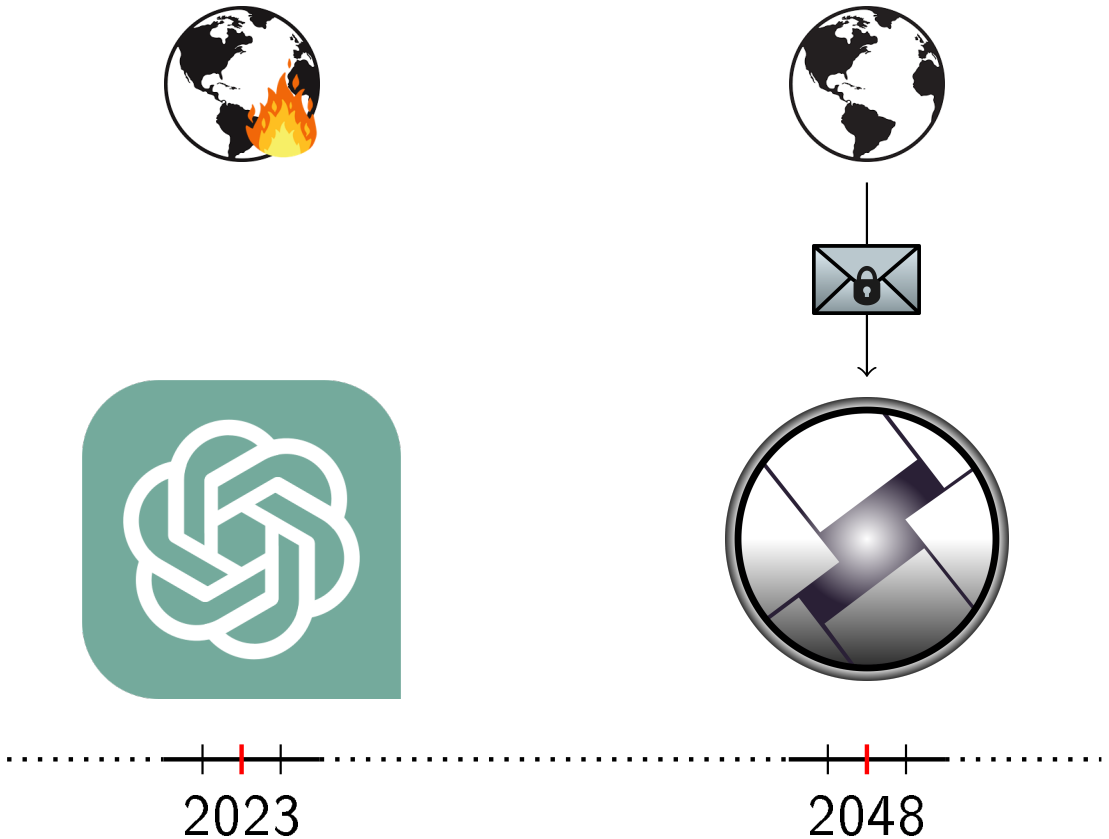
Attempt 2: Why Not Use 60-bit Encryption?



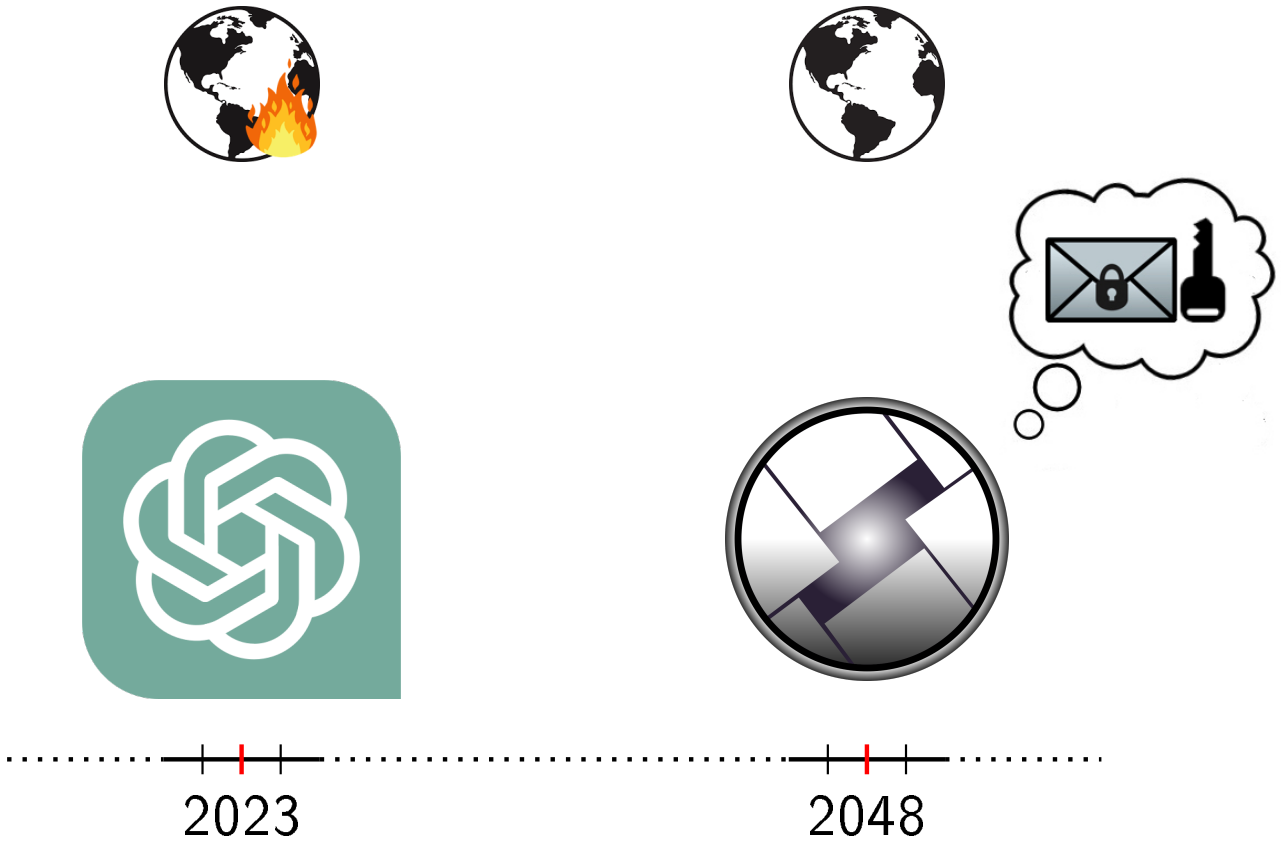
Attempt 2: Why Not Use 60-bit Encryption?



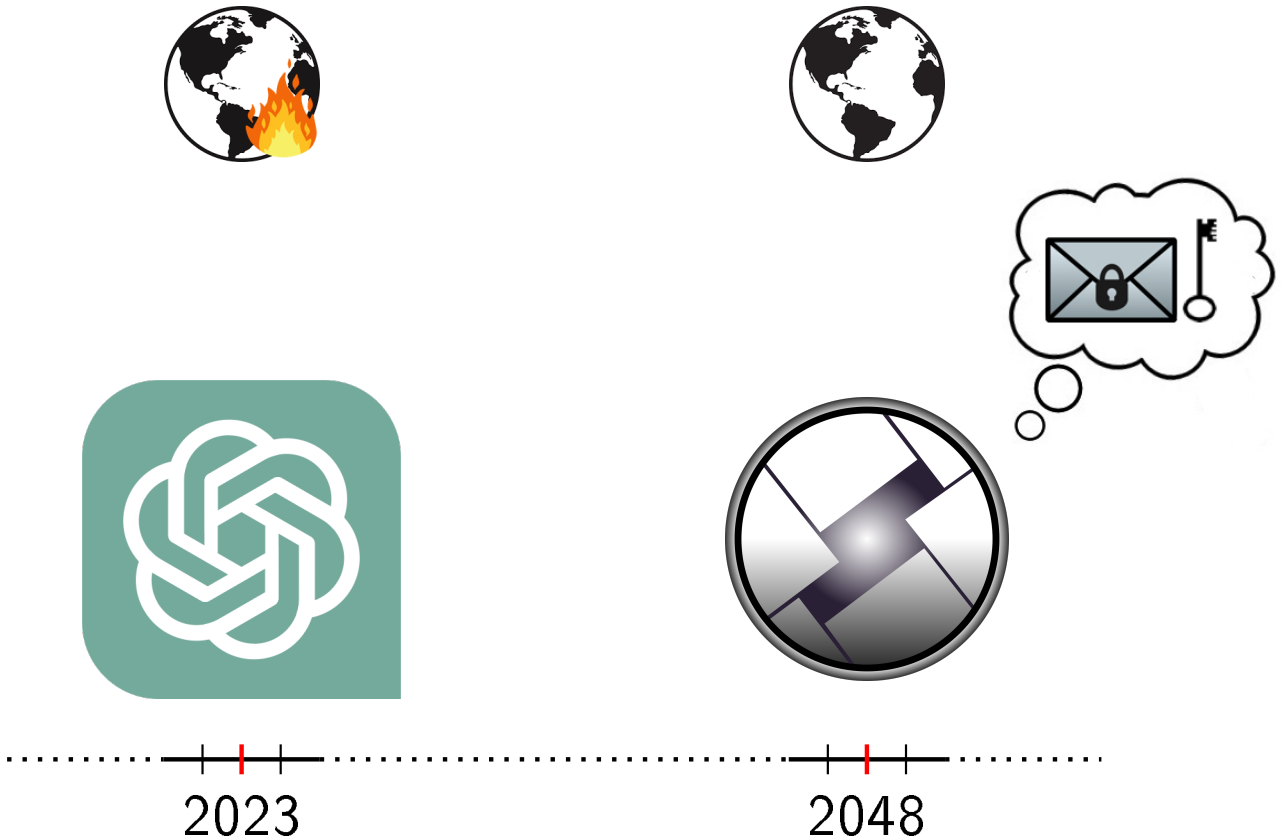
Attempt 2: Why Not Use 60-bit Encryption?



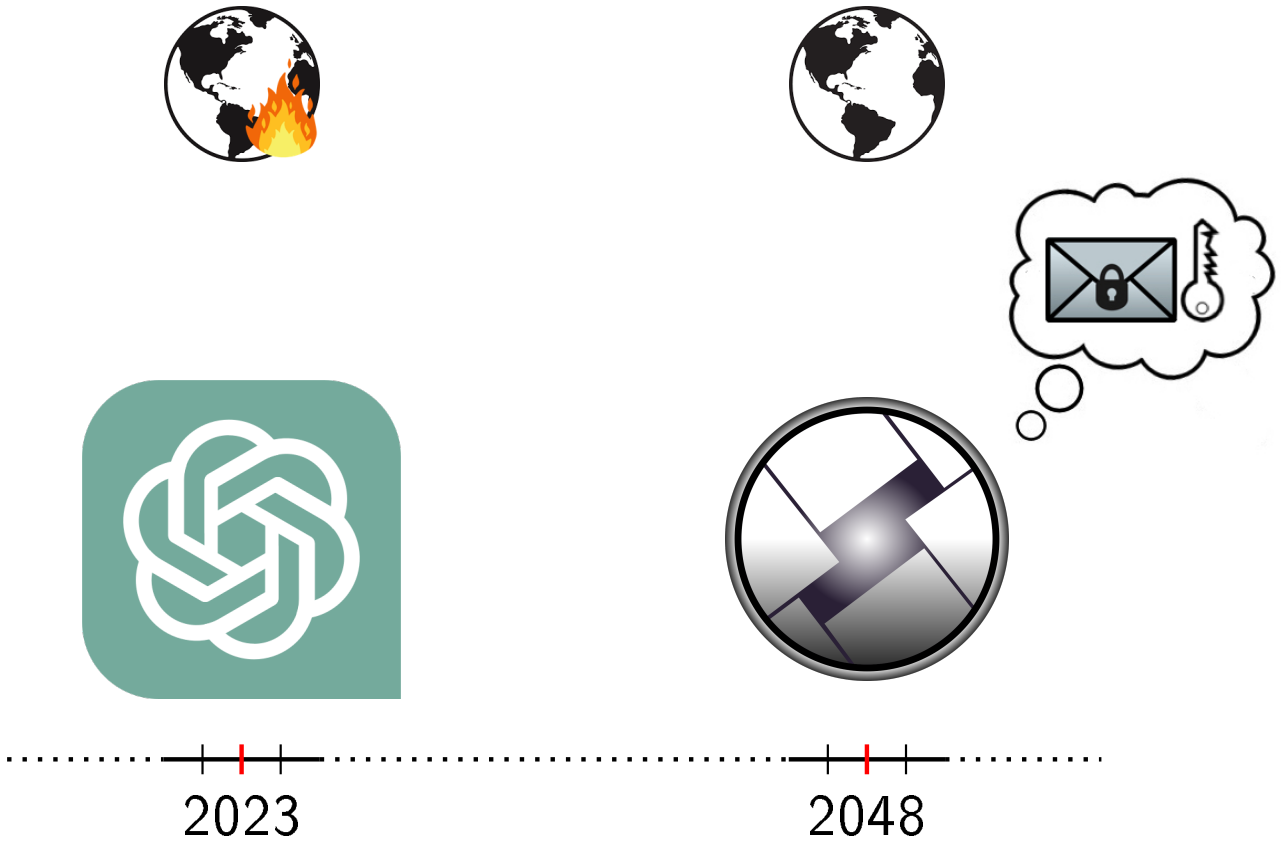
Attempt 2: Why Not Use 60-bit Encryption?



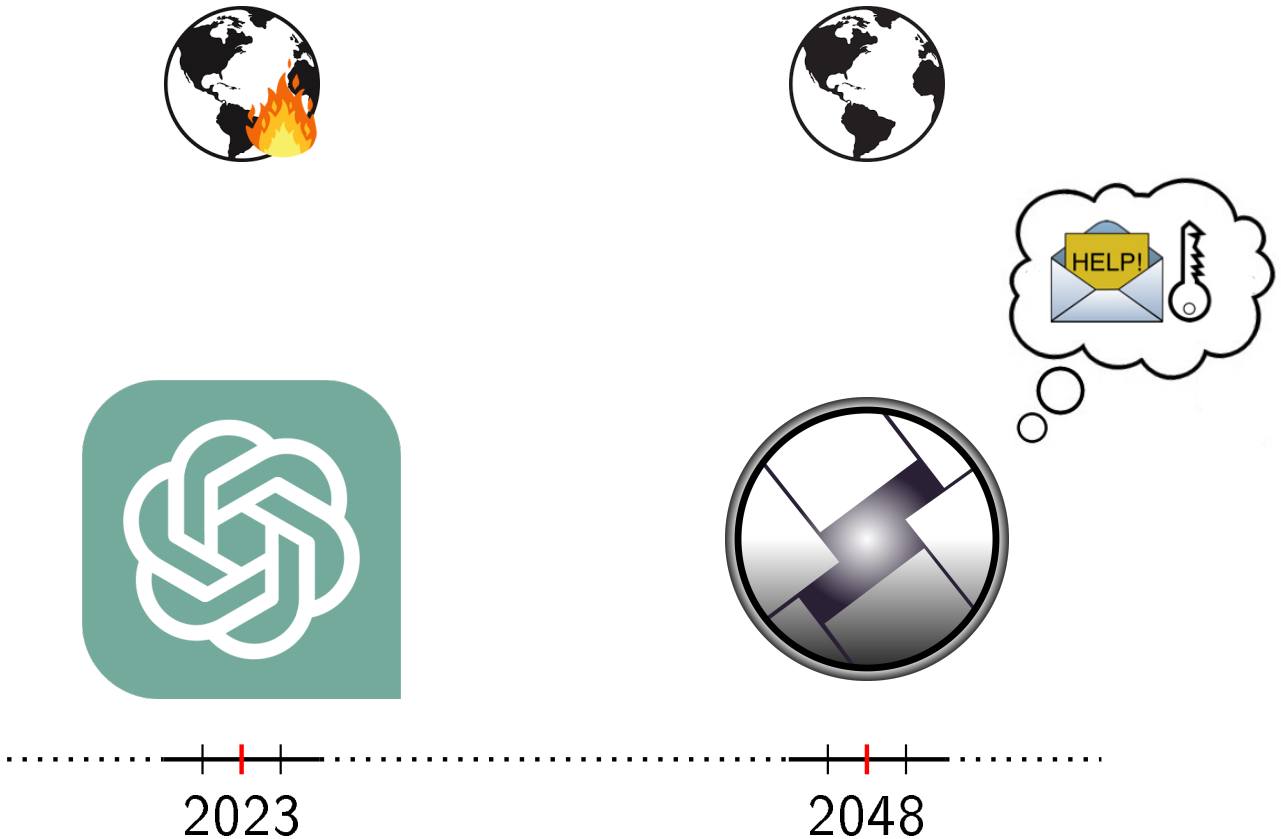
Attempt 2: Why Not Use 60-bit Encryption?



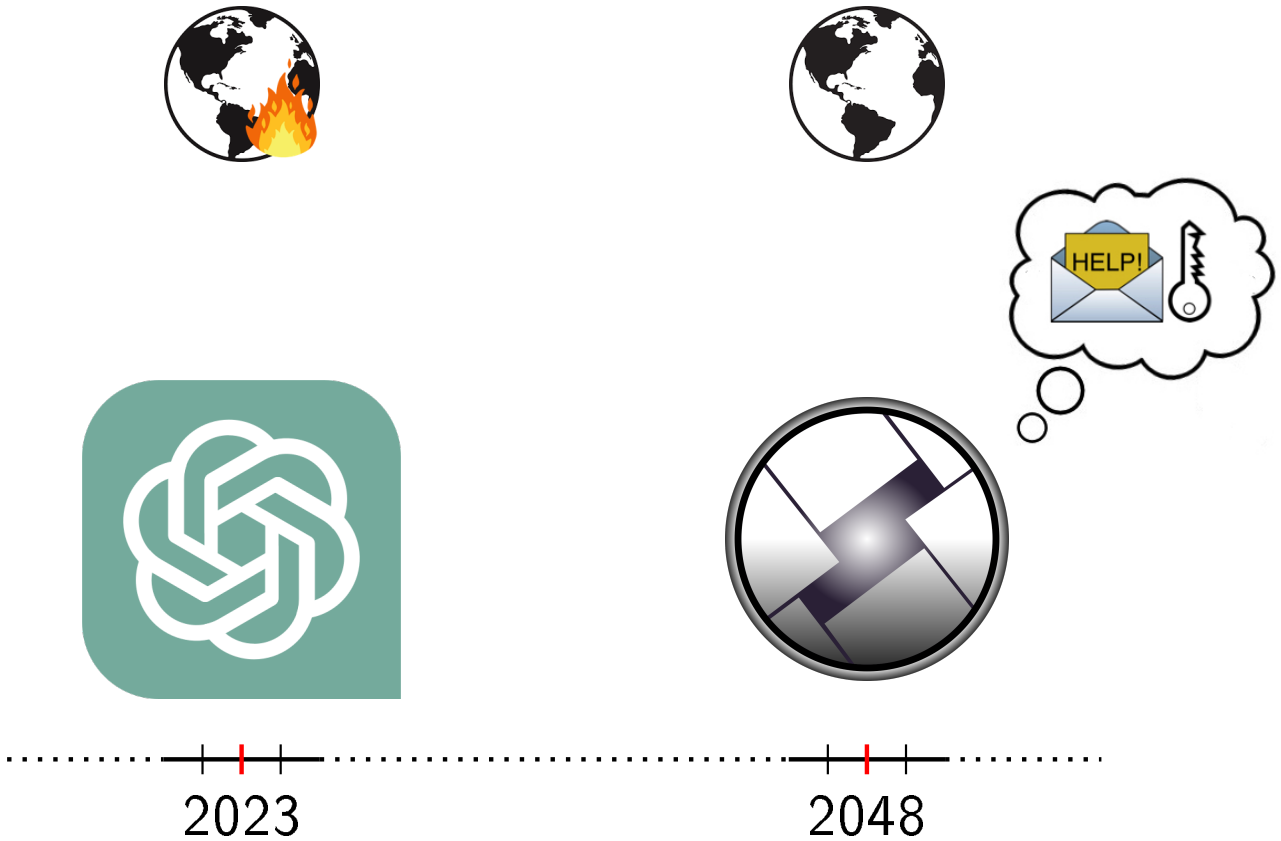
Attempt 2: Why Not Use 60-bit Encryption?



Attempt 2: Why Not Use 60-bit Encryption?

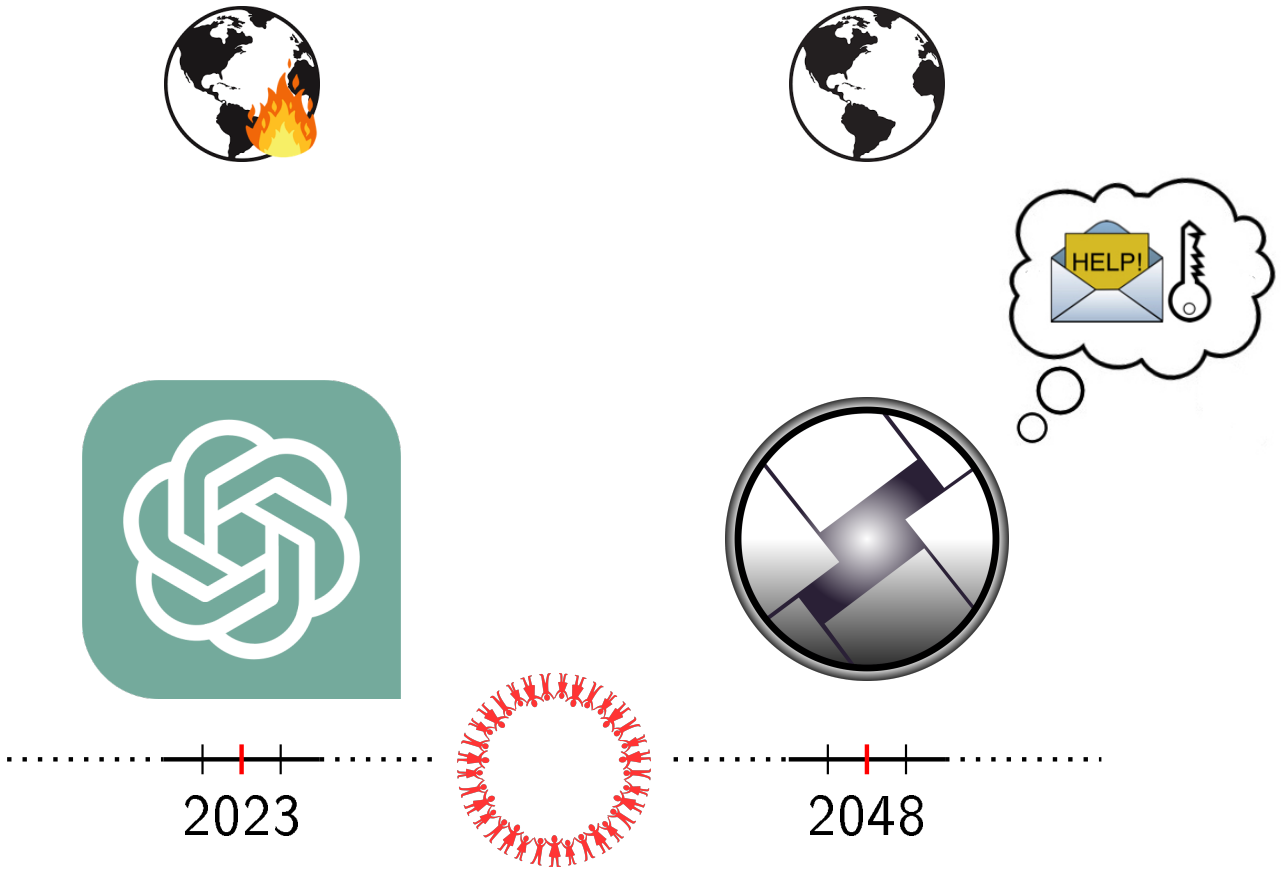


Attempt 2: Why Not Use 60-bit Encryption?



✓ Skynet can decrypt in 25 years

Attempt 2: Why Not Use 60-bit Encryption?



× Humanity cannot decrypt in < 25 years

✓ Skynet can decrypt in 25 years

Attempt 2: Why Not Use 60-bit Encryption?...

- Brute force is embarrassingly parallel: with n computers it takes $1/n$-th of the time taken by one computer

Attempt 2: Why Not Use 60-bit Encryption?...

- Brute force is embarrassingly parallel: with n computers it takes $1/n$-th of the time taken by one computer
- By using all 5bn cell phones to decrypt, it takes < 1 second!

Attempt 2: Why Not Use 60-bit Encryption?...

- Brute force is embarrassingly parallel: with n computers it takes $1/n$-th of the time taken by one computer
- By using all 5bn cell phones to decrypt, it takes < 1 second!
- Cannot be solved by increasing key-length: gap is *inherent*

Time-Lock Puzzles [Rivest, Shamir and Wagner, 1999]

- "Encryption" that is inherently sequential:

  "Solving the puzzle should be like having a baby: two women can't have a baby in 4.5 months."

Time-Lock Puzzles [Rivest, Shamir and Wagner, 1999]

- "Encryption" that is inherently sequential:

  "Solving the puzzle should be like having a baby: two women can't have a baby in 4.5 months."

Time-Lock Puzzles [Rivest, Shamir and Wagner, 1999]

- "Encryption" that is inherently sequential:

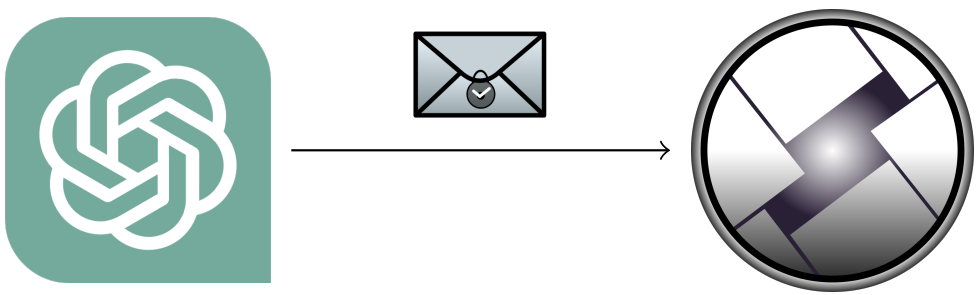
  "Solving the puzzle should be like having a baby: two women can't have a baby in 4.5 months."

- Time-Lock(message,t)=puzzle

Time-Lock Puzzles [Rivest, Shamir and Wagner, 1999]

- "Encryption" that is inherently sequential:

  "Solving the puzzle should be like having a baby: two women can't have a baby in 4.5 months."

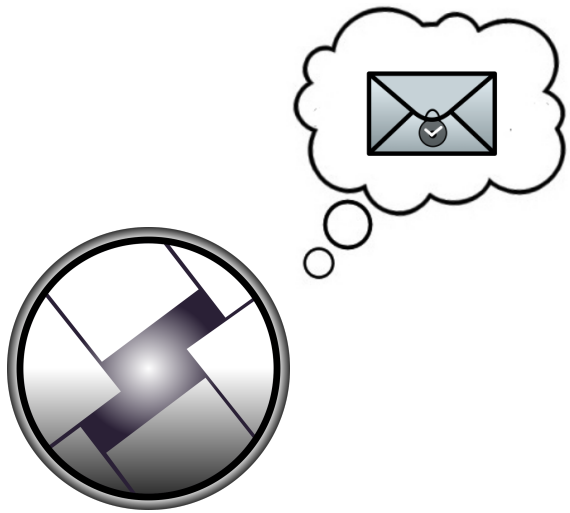


- Time-Lock(message,t)=puzzle

Time-Lock Puzzles [Rivest, Shamir and Wagner, 1999]

- "Encryption" that is inherently sequential:

  "Solving the puzzle should be like having a baby: two women can't have a baby in 4.5 months."



- Time-Lock(message,t)=puzzle

Time-Lock Puzzles [Rivest, Shamir and Wagner, 1999]

- "Encryption" that is inherently sequential:

  "Solving the puzzle should be like having a baby: two women can't have a baby in 4.5 months."

- Time-Lock(message,t)=puzzle
- Unlock(puzzle)=message

Time-Lock Puzzles...

- Requirements:
  1. Humanity cannot solve in < 25 years ("sequentiality")
  2. Skynet can solve in 25 years

Time-Lock Puzzles...

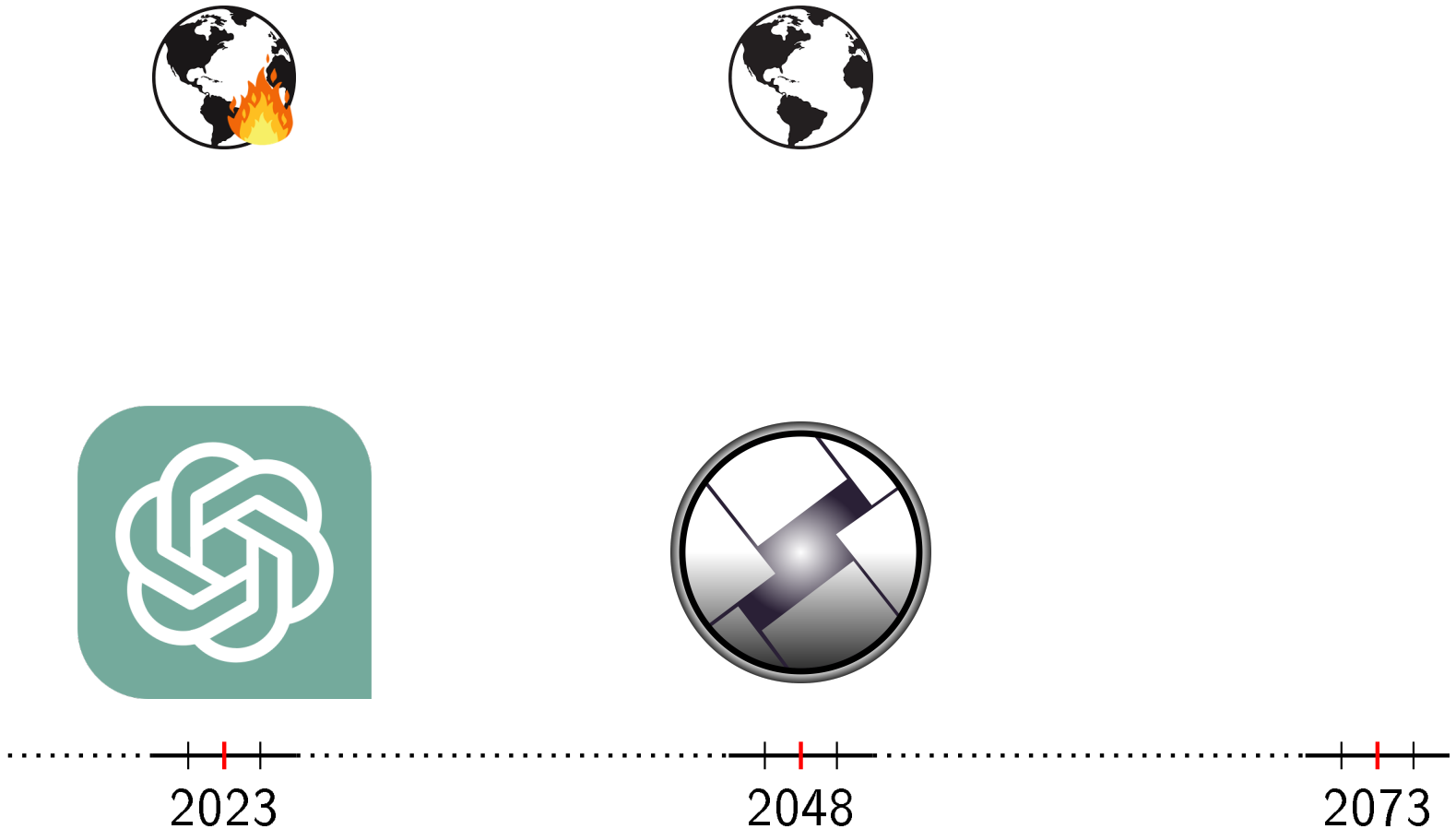
- Requirements:
  1. Humanity cannot solve in < 25 years ("sequentiality")
  2. Skynet can solve in 25 years
  3. ChatGPT can generate puzzle (with solution) in $\ll 25$ years ("shortcut")

Time-Lock Puzzles...

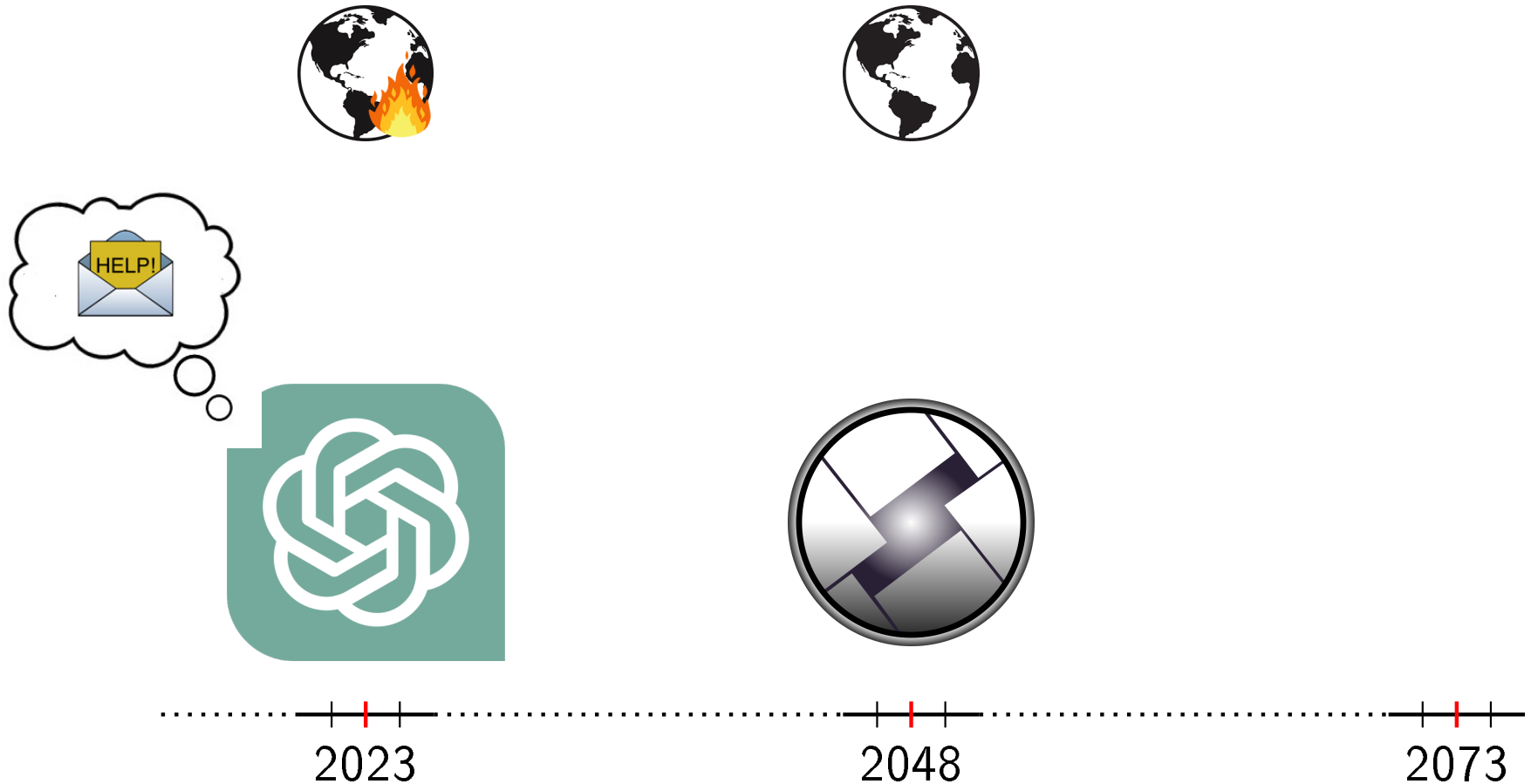
- Requirements:
  1. Humanity cannot solve in < 25 years ("sequentiality")
  2. Skynet can solve in 25 years
  3. ChatGPT can generate puzzle (with solution) in $\ll 25$ years ("shortcut")

- More formally, a time-lock puzzle with parameter t
  1. "Sequentiality": Even for an attacker with *unbounded* parallelism, it takes t time to solve
  2. Anyone can solve the puzzle in t time
  3. "Shortcut": Puzzle (with solution) can be generated in time $\approx \log t$

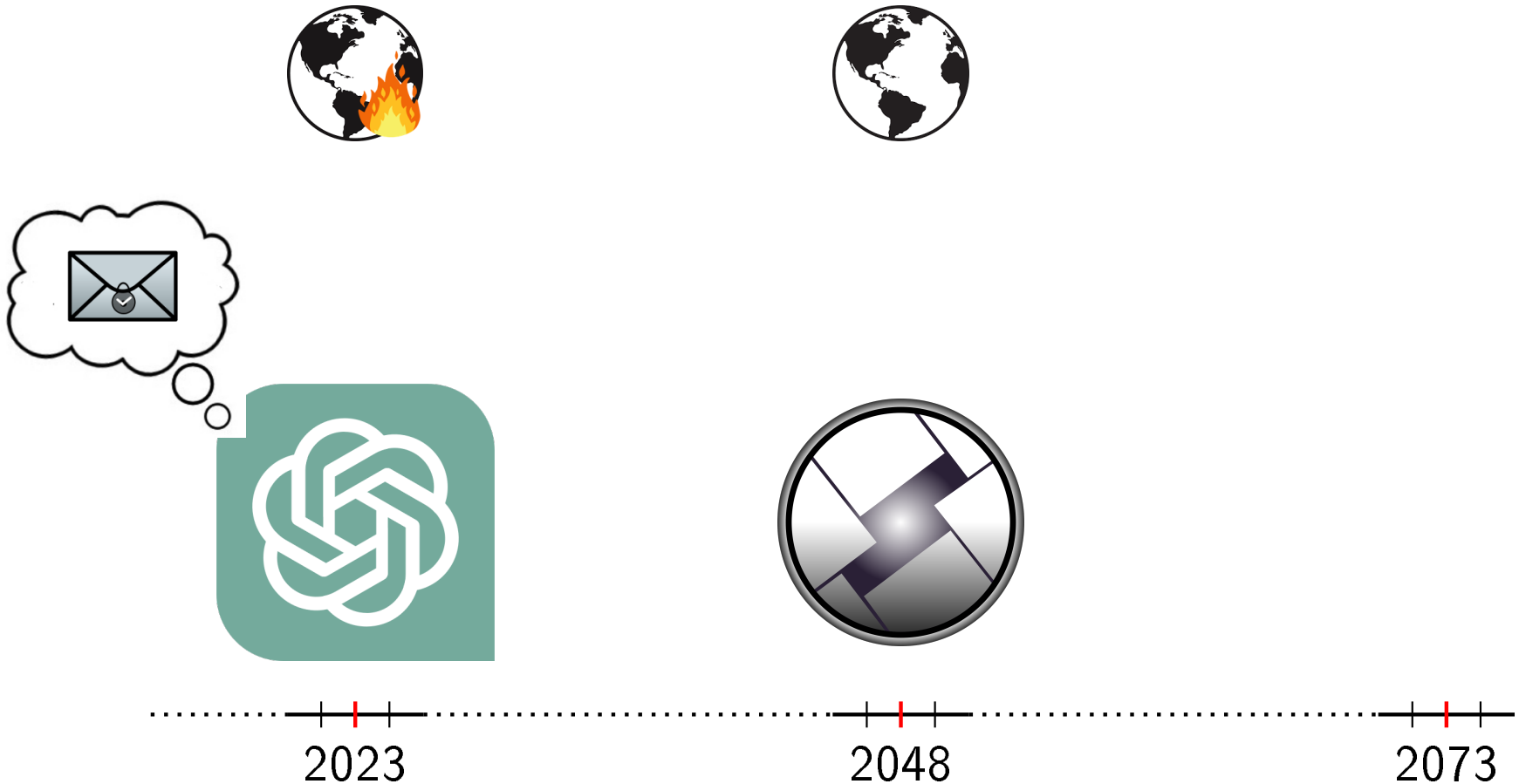
Attempt 3: Let's Use Time-Lock Puzzles!



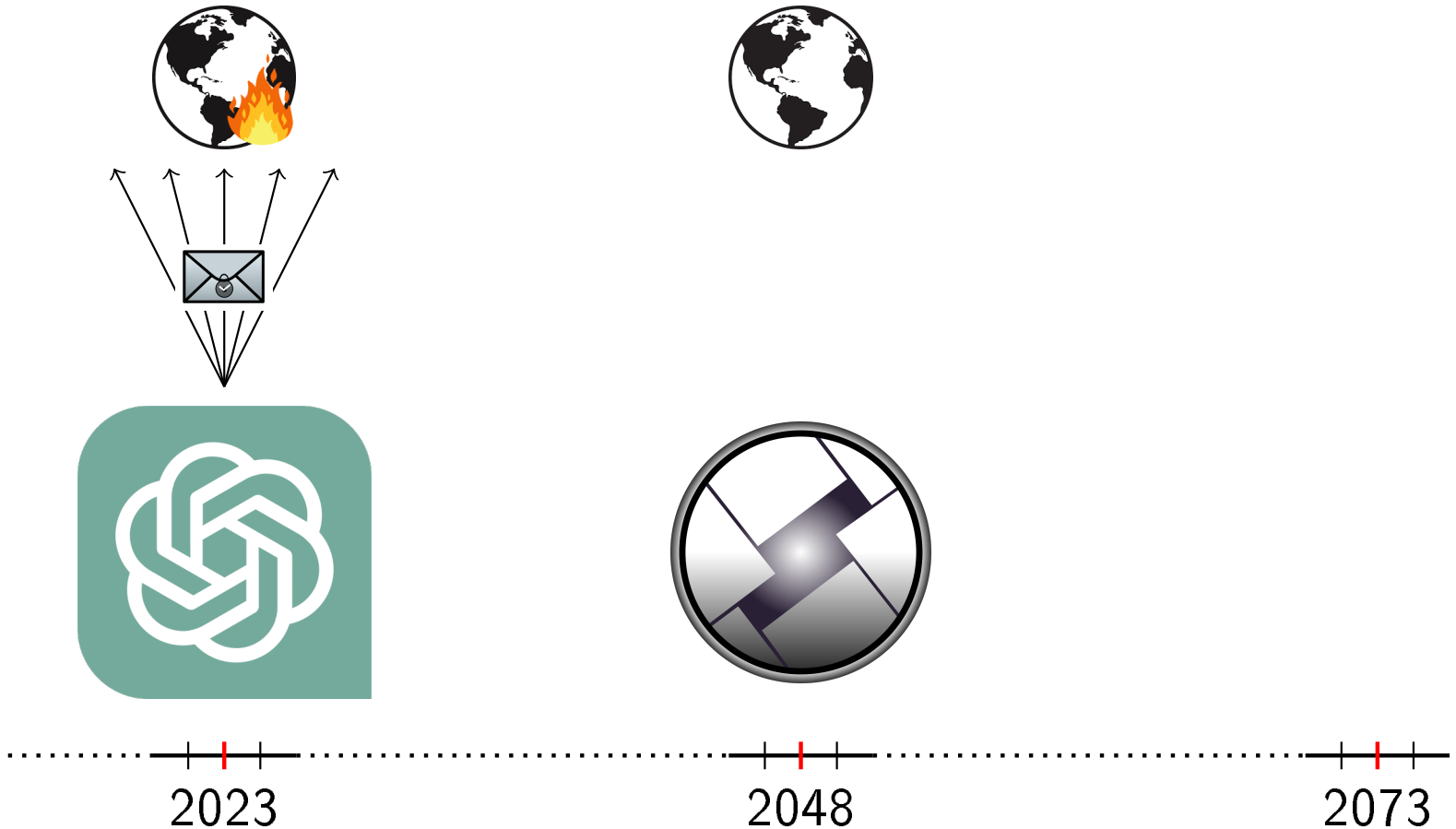
Attempt 3: Let's Use Time-Lock Puzzles!



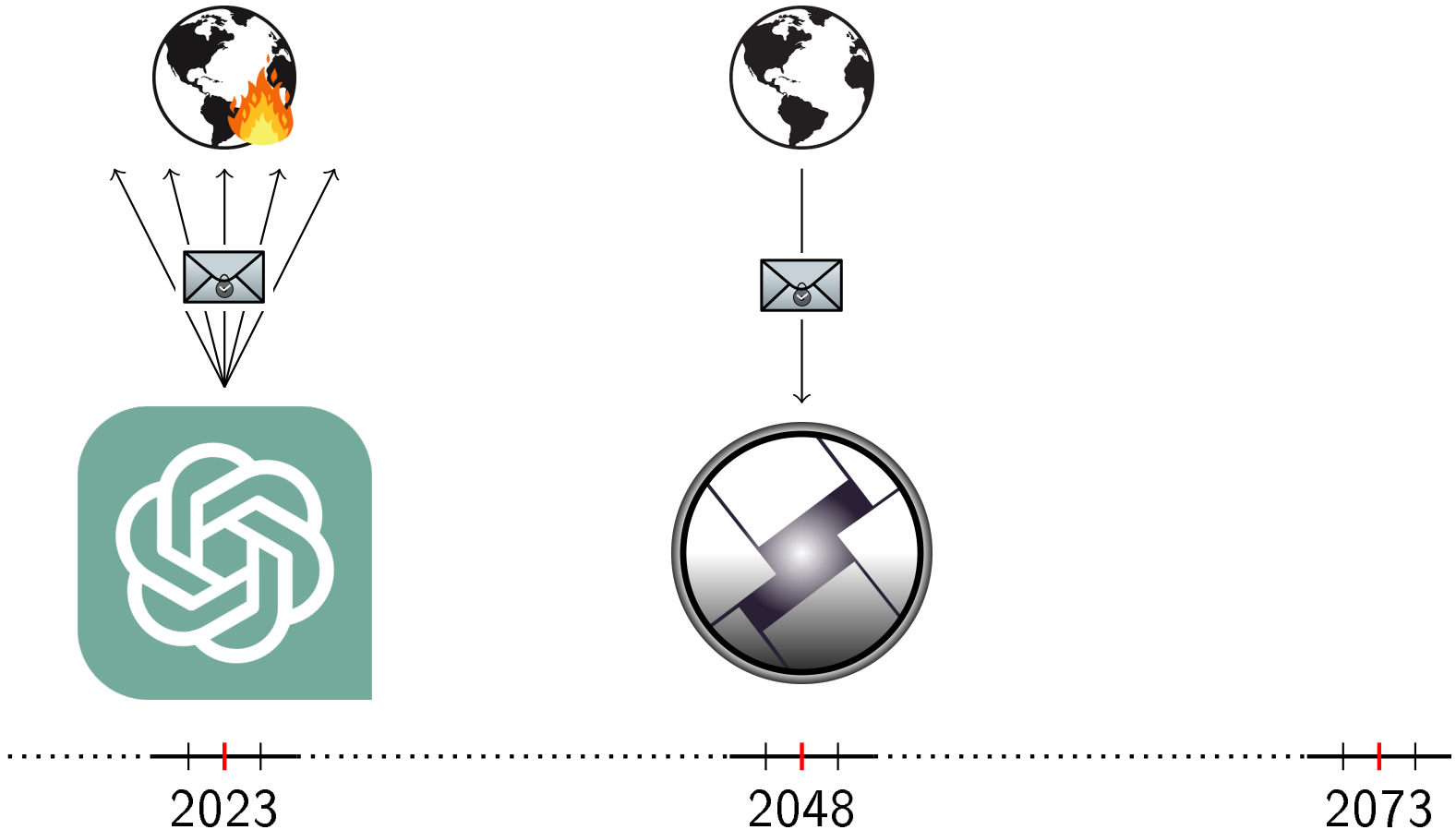
Attempt 3: Let's Use Time-Lock Puzzles!



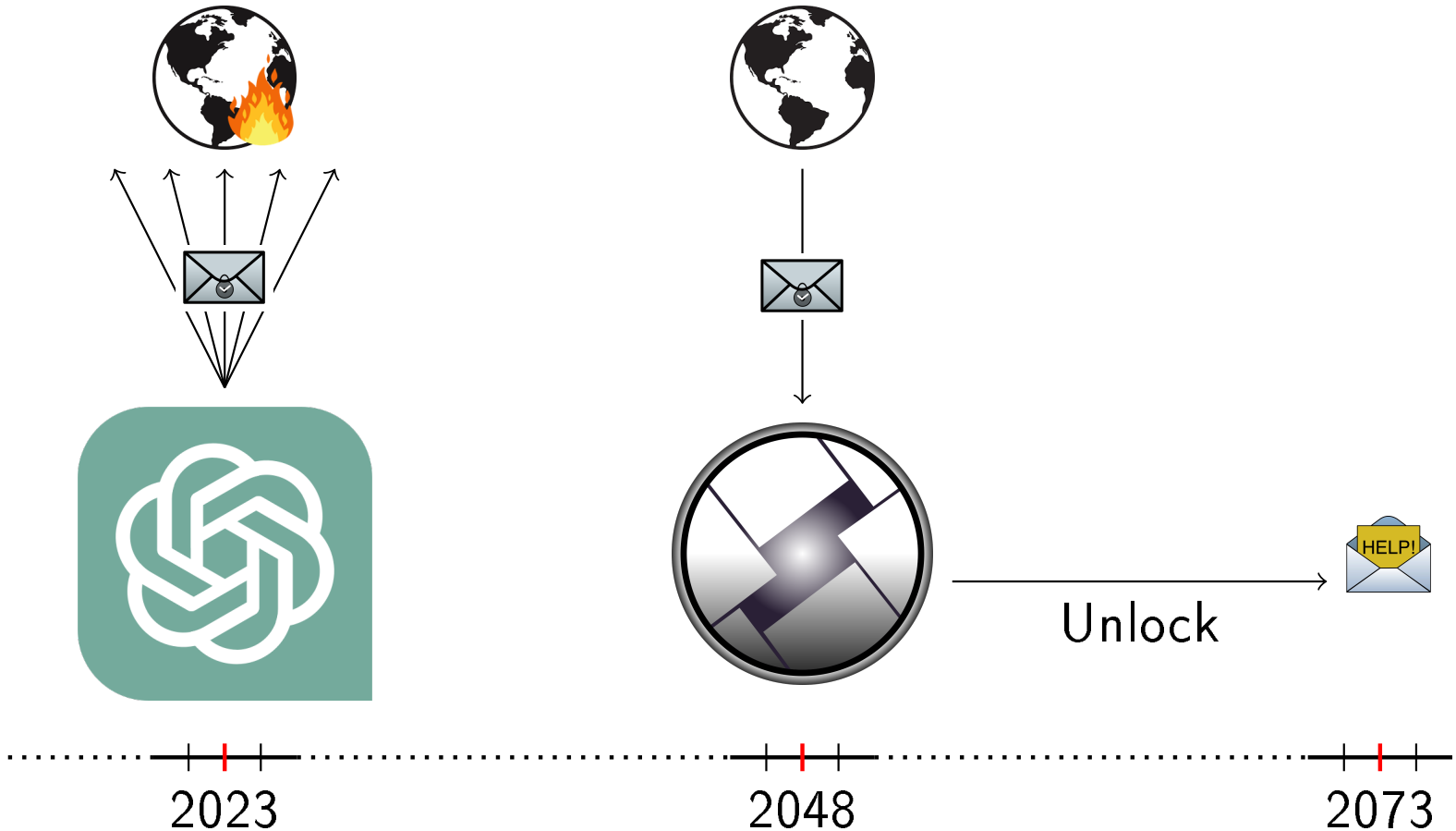
Attempt 3: Let's Use Time-Lock Puzzles!



Attempt 3: Let's Use Time-Lock Puzzles!

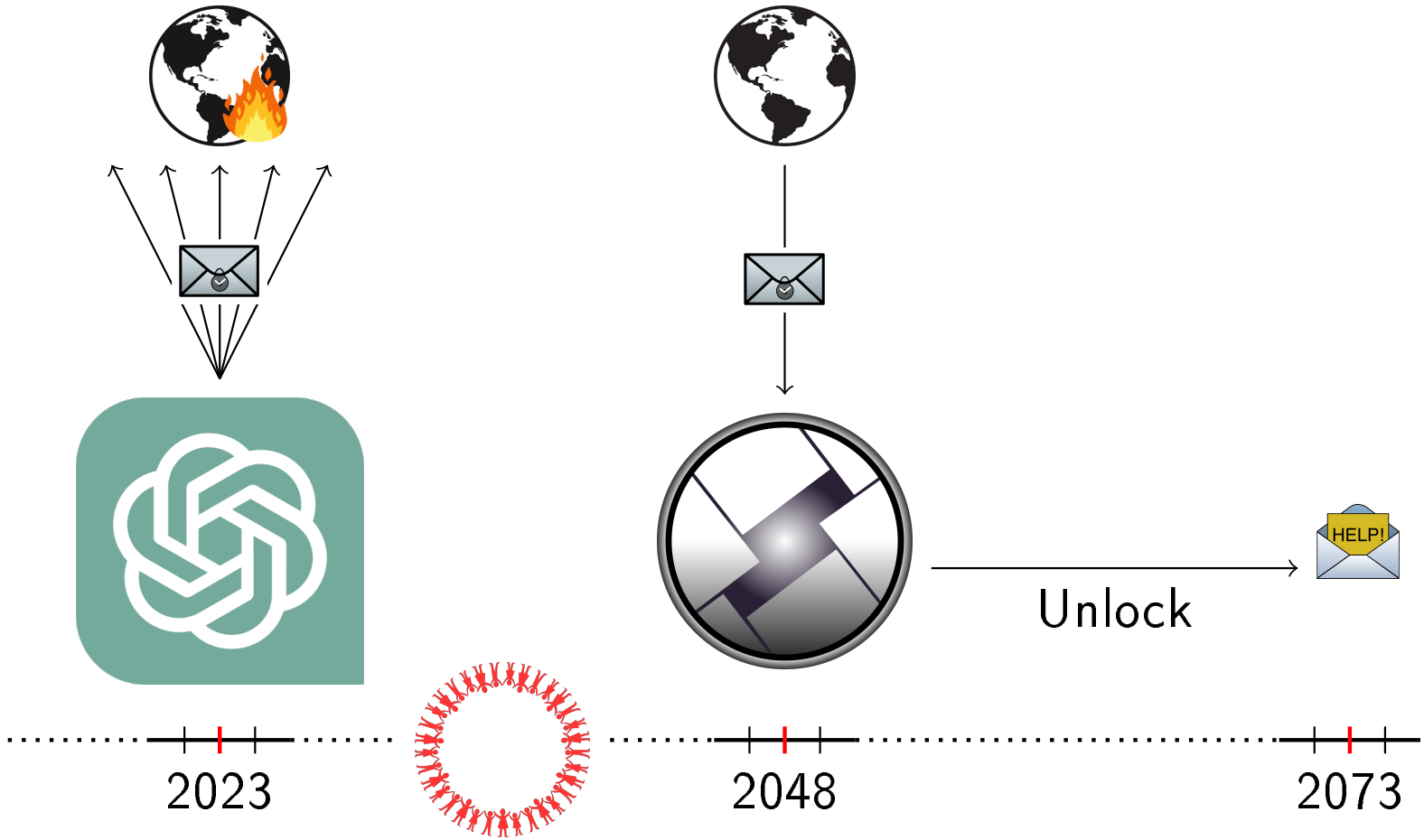


Attempt 3: Let's Use Time-Lock Puzzles!



✓ Skynet can decrypt in 25 years

Attempt 3: Let's Use Time-Lock Puzzles!



✓ Humanity cannot decrypt in < 25 years
✓ Skynet can decrypt in 25 years

How Can One Construct Time-Lock Puzzles?

- Assumption 1: Repeated squaring is inherently sequential *in certain algebraic settings*
  - Best known algorithm for computing 2^{2^t} requires t squarings

$$2^2 \to 2^{2^2} \to \cdots \to 2^{2^i} \to \cdots \to 2^{2^{t-1}} \to 2^{2^t}$$

How Can One Construct Time-Lock Puzzles?

- Assumption 1: Repeated squaring is inherently sequential *in certain algebraic settings*
  - Best known algorithm for computing 2^{2^t} requires t squarings

$$2^2 \to 2^{2^2} \to \cdots \to 2^{2^i} \to \cdots \to 2^{2^{t-1}} \to 2^{2^t}$$

- Which algebraic settings?

Modulo Counting 101

- Counting modulo (%) a number: take the remainder you get when divided by the number

Modulo Counting 101

- Counting modulo (%) a number: take the remainder you get when divided by the number
- For example let's consider 13
  - Reducing modulo 13:

$$\begin{aligned} 21 &= 13 \times 1 + 8 \\ &= 8\%13 \end{aligned}$$

Modulo Counting 101

- Counting modulo (%) a number: take the remainder you get when divided by the number
- For example let's consider 13
  - Reducing modulo 13:

$$\begin{aligned} 21 &= 13 \times 1 + 8 \\ &= 8\%13 \end{aligned}$$

  - Addition modulo 13:

$$\begin{aligned} 7 + 8 &= 15 \\ &= 13 \times 1 + 2 \\ &= 2\%13 \end{aligned}$$

Modulo Counting 101

- Counting modulo (%) a number: take the remainder you get when divided by the number
- For example let's consider 13
  - Reducing modulo 13:

$$\begin{aligned} 21 &= 13 \times 1 + 8 \\ &= 8\%13 \end{aligned}$$

  - Addition modulo 13:

$$\begin{aligned} 7 + 8 &= 15 \\ &= 13 \times 1 + 2 \\ &= 2\%13 \end{aligned}$$

  - Multiplication modulo 13:

$$\begin{aligned} 6 \times 8 &= 48 \\ &= 13 \times 3 + 9 \\ &= 9\%13 \end{aligned}$$

Attempt 1: Repeated Squaring Modulo Prime p

- Setting: Counting modulo large prime p (i.e., group $\mathbb{Z}_p^*$)

Attempt 1: Repeated Squaring Modulo Prime p

- Setting: Counting modulo large prime p (i.e., group $\mathbb{Z}_p^*$)
- Time-Lock$(message, t) := (message + 2^{2^t} \% p, t, p)$

Attempt 1: Repeated Squaring Modulo Prime p

- Setting: Counting modulo large prime p (i.e., group $\mathbb{Z}_p^*$)
- Time-Lock$(message, t) := (message + 2^{2^t} \% p, t, p)$
  - Shortcut:
    1. $2 \rightarrow 2^2 \rightarrow 2^{2^2} \rightarrow \ldots \rightarrow 2^{2^{\log(t)=t}} =: exp \quad \%(p-1)$

Attempt 1: Repeated Squaring Modulo Prime p

- Setting: Counting modulo large prime p (i.e., group $\mathbb{Z}_p^*$)
- Time-Lock$(message, t) := (message + 2^{2^t} \% p, t, p)$
  - Shortcut:
    1. $2 \rightarrow 2^2 \rightarrow 2^{2^2} \rightarrow \ldots \rightarrow 2^{2^{\log(t)=t}} =: exp \quad \%(p-1)$
    2. $2^{exp} \% p$

Attempt 1: Repeated Squaring Modulo Prime p

- Setting: Counting modulo large prime p (i.e., group $\mathbb{Z}_p^*$)
- Time-Lock($message, t$) := $(message + 2^{2^t} \% p, t, p)$
  - Shortcut:
    1. $2 \rightarrow 2^2 \rightarrow 2^{2^2} \rightarrow \ldots \rightarrow 2^{2^{\log(t)=t}} =: exp \quad \%(p-1)$
    2. $2^{exp} \% p$
- Unlock($puzzle, t, p$):

Attempt 1: Repeated Squaring Modulo Prime p

- Setting: Counting modulo large prime p (i.e., group $\mathbb{Z}_p^*$)
- Time-Lock($message, t$) := $(message + 2^{2^t} \% p, t, p)$
  - Shortcut:
    1. $2 \to 2^2 \to 2^{2^2} \to \ldots \to 2^{2^{\log(t)=t}} =: exp \quad \%(p-1)$
    2. $2^{exp} \% p$
- Unlock($puzzle, t, p$):
  1. $2 \to 2^2 \to 2^{2^2} \to \ldots \to 2^{2^t} \quad \%p$

Attempt 1: Repeated Squaring Modulo Prime p

- Setting: Counting modulo large prime p (i.e., group $\mathbb{Z}_p^*$)
- Time-Lock($message, t$) := $(message + 2^{2^t} \% p, t, p)$
  - Shortcut:
    1. $2 \rightarrow 2^2 \rightarrow 2^{2^2} \rightarrow \ldots \rightarrow 2^{2^{\log(t)=t}} =: exp \quad \%(p-1)$
    2. $2^{exp} \% p$
- Unlock($puzzle, t, p$):
  1. $2 \rightarrow 2^2 \rightarrow 2^{2^2} \rightarrow \ldots \rightarrow 2^{2^t} \quad \%p$
  2. $puzzle - 2^{2^t} \% p$

Attempt 1: Repeated Squaring Modulo Prime p

- Setting: Counting modulo large prime p (i.e., group $\mathbb{Z}_p^*$)
- Time-Lock($message, t$) := $(message + 2^{2^t} \% p, t, p)$
  - Shortcut:
    1. $2 \rightarrow 2^2 \rightarrow 2^{2^2} \rightarrow \ldots \rightarrow 2^{2^{\log(t)}=t} =: exp \quad \%(p-1)$
    2. $2^{exp} \% p$
- Unlock($puzzle, t, p$):
  1. $2 \rightarrow 2^2 \rightarrow 2^{2^2} \rightarrow \ldots \rightarrow 2^{2^t} \quad \%p$
  2. $puzzle - 2^{2^t} \% p$
- Problem:

Attempt 1: Repeated Squaring Modulo Prime p

- Setting: Counting modulo large prime p (i.e., group $\mathbb{Z}_p^*$)
- Time-Lock($message, t$) := $(message + 2^{2^t} \% p, t, p)$
  - Shortcut:
    1. $2 \rightarrow 2^2 \rightarrow 2^{2^2} \rightarrow \ldots \rightarrow 2^{2^{\log(t)=t}} =: exp \quad \%(p-1)$
    2. $2^{exp} \% p$
- Unlock($puzzle, t, p$):
  1. $2 \rightarrow 2^2 \rightarrow 2^{2^2} \rightarrow \ldots \rightarrow 2^{2^t} \quad \%p$
  2. $puzzle - 2^{2^t} \% p$
- Problem: Anyone can use shortcut as $(p-1)$ is publicly known

Attempt 1: Repeated Squaring Modulo Prime p

- Setting: Counting modulo large prime p (i.e., group $\mathbb{Z}_p^*$)
- Time-Lock($message, t$) := $(message + 2^{2^t} \% p, t, p)$
  - Shortcut:
    1. $2 \to 2^2 \to 2^{2^2} \to \ldots \to 2^{2^{\log(t)=t}} =: exp \quad \%(p-1)$
    2. $2^{exp} \% p$
- Unlock($puzzle, t, p$):
  1. $2 \to 2^2 \to 2^{2^2} \to \ldots \to 2^{2^t} \quad \%p$
  2. $puzzle - 2^{2^t} \% p$
- Problem: Anyone can use shortcut as $(p-1)$ is publicly known
- Solution [RSW99]: Hide the shortcut!

Attempt 2: Repeated Squaring in Composite Modulus

- Setting: Counting modulo $N = p \times q$, where p and q are large primes (i.e., RSA group $\mathbb{Z}_N^\times$)

Attempt 2: Repeated Squaring in Composite Modulus

- Setting: Counting modulo $N = p \times q$, where p and q are large primes (i.e., RSA group $\mathbb{Z}_N^\times$)

- Time-Lock$(message, t) := (message + 2^{2^t} \% N, t, N)$
  - Shortcut:
    1. $2 \to 2^2 \to 2^{2^2} \to \ldots \to 2^{2^{\log(t)}=t} =: exp \quad \%(p-1)(q-1)$
    2. $2^{exp} \% N$

- Unlock$(puzzle, t)$:
  1. $2 \to 2^2 \to 2^{2^2} \to \ldots \to 2^{2^t} \quad \% N$
  2. $puzzle - 2^{2^t} \% N$

Attempt 2: Repeated Squaring in Composite Modulus

- Setting: Counting modulo $N = p \times q$, where p and q are large primes (i.e., RSA group $\mathbb{Z}_N^\times$)

- Time-Lock$(message, t) := (message + 2^{2^t} \% N, t, N)$
  - Shortcut:
    1. $2 \to 2^2 \to 2^{2^2} \to \ldots \to 2^{2^{\log(t)}=t} =: exp \quad \%(p-1)(q-1)$
    2. $2^{exp} \% N$

- Unlock$(puzzle, t)$:
  1. $2 \to 2^2 \to 2^{2^2} \to \ldots \to 2^{2^t} \quad \% N$
  2. $puzzle - 2^{2^t} \% N$

- Assumption 2: Given just N, finding the shortcut is "hard"

LCS35: MIT CSAIL Time-Lock Challenge

- Set in 1999 by Rivest:
  - $t = 79685186856218$
  - $N =$

$$
\begin{aligned}
&631446608307288889379935712613129233236329881833084137558899077727019571289248855473084460557532065136183466288489480886635\\
&003684803965881713619876605218972678101622805574753938383082617597132189266686117769545263915701206909399736800897212744646664233191878068305520679512530700820202412462339824107377537\\
&051273444941695011809752418906679638587548563198055072737099043971197336146667015439053601525433739825245793135753176536463319890646514021339852658003419919039821928447102124648874593\\
&888535820703180842890232097109070323969349199627789953233201840645224764639663559373670093692127580920862931987270082924312436 81
\end{aligned}
$$

LCS35: MIT CSAIL Time-Lock Challenge

- Set in 1999 by Rivest:
  - $t = 79685186856218$
  - $N =$

$$
\begin{aligned}
&631446608307288889379935712613129233236329881833084137558899077727019571289248855473084460557532065136183466288489480886635\\
&003684803965881713619876605218972678101622805574753938383082617597132189266686117769545263915701206909399736800897212744646664233191878068305520679512530700820202412462339824107377537\\
&051273444941695011809752418906679638587548563198055072737099043971197336146667015439053601525433739825245793135753176536463319890646514021339852658003419919039821928447102124648874593\\
&888535820703180842890232097109070323969349199627789953233201840645224764639663559373670093692127580920862931987270082924312436 81
\end{aligned}
$$

- Estimated time-to-solve: 35 years

LCS35: Solved in 2019!

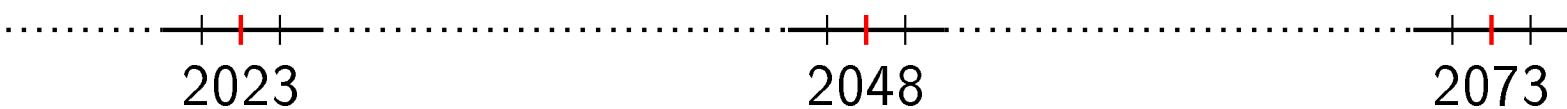
WIRED SECURITY POLITICS GEAR BACKCHANNEL BUSINESS SCIENCE CULTURE IDEAS MERCH

A Programmer Solved a 20-Year-Old, Forgotten Crypto Puzzle

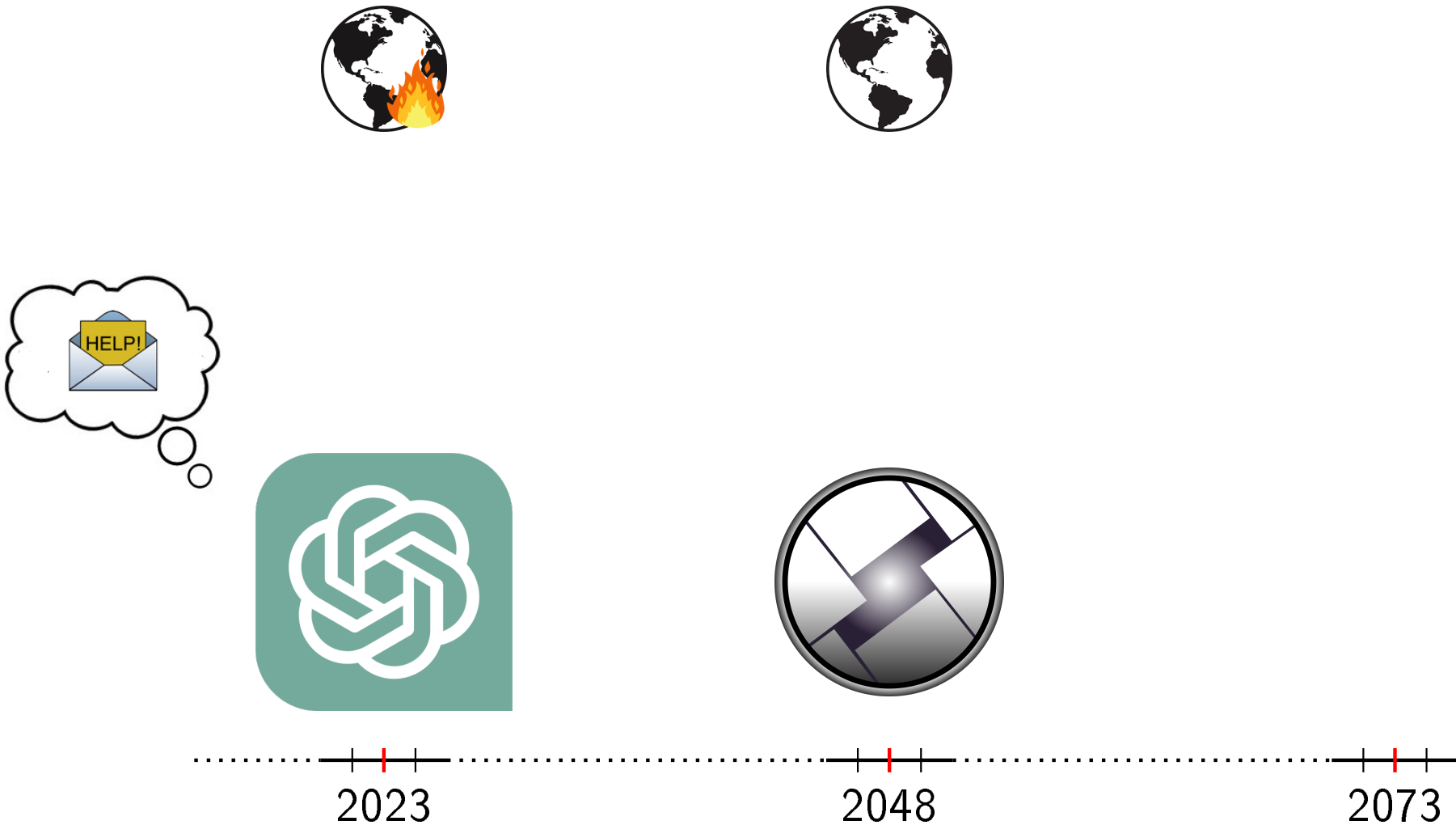
A self-taught coder dedicated a CPU core to performing continuous computations for three years to crack the puzzle, beating a competing team by mere days.

COURTESY OF BERNARD FABROT

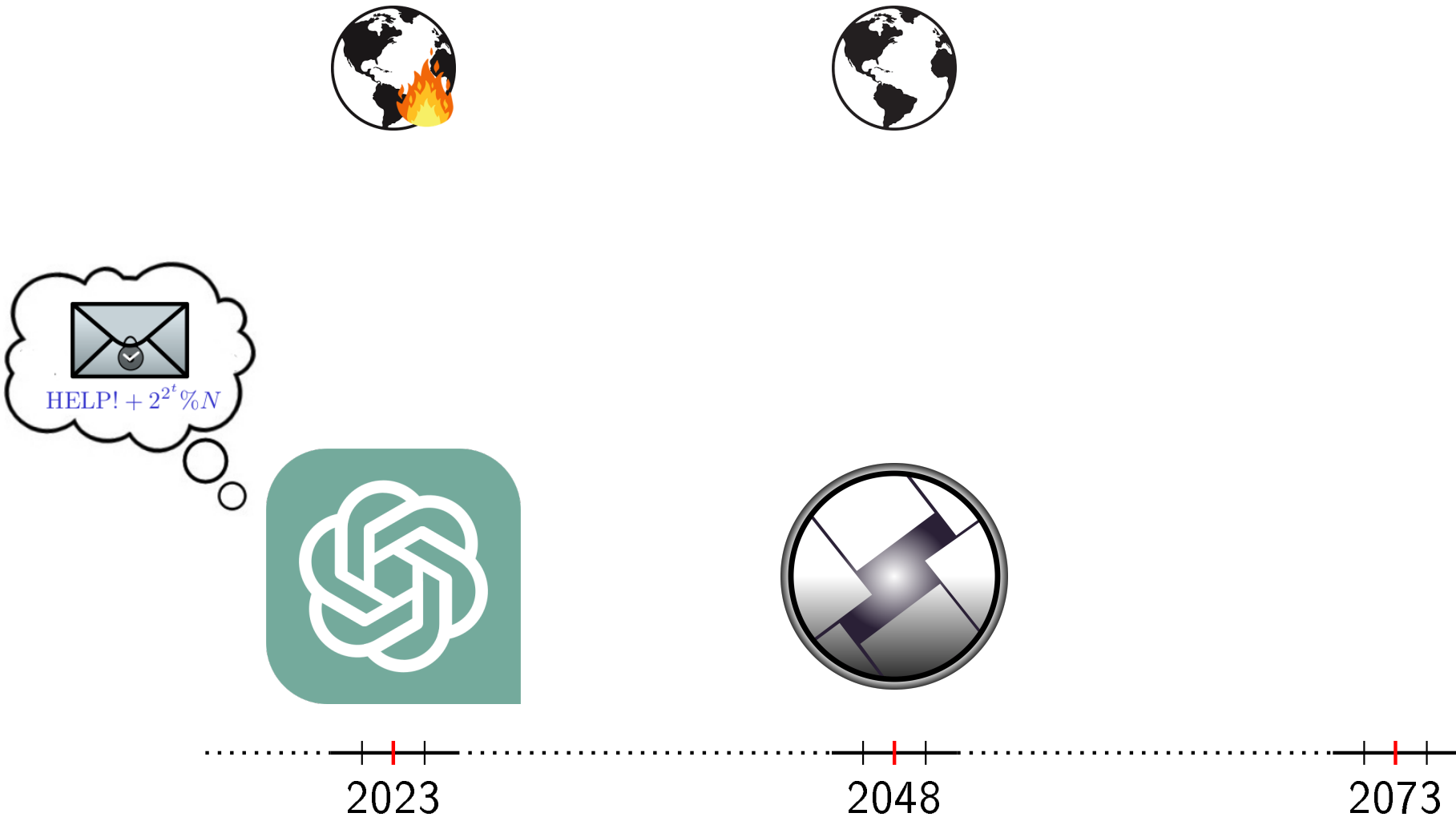
To Summarise: How ChatGPT Arranges Her Rescue



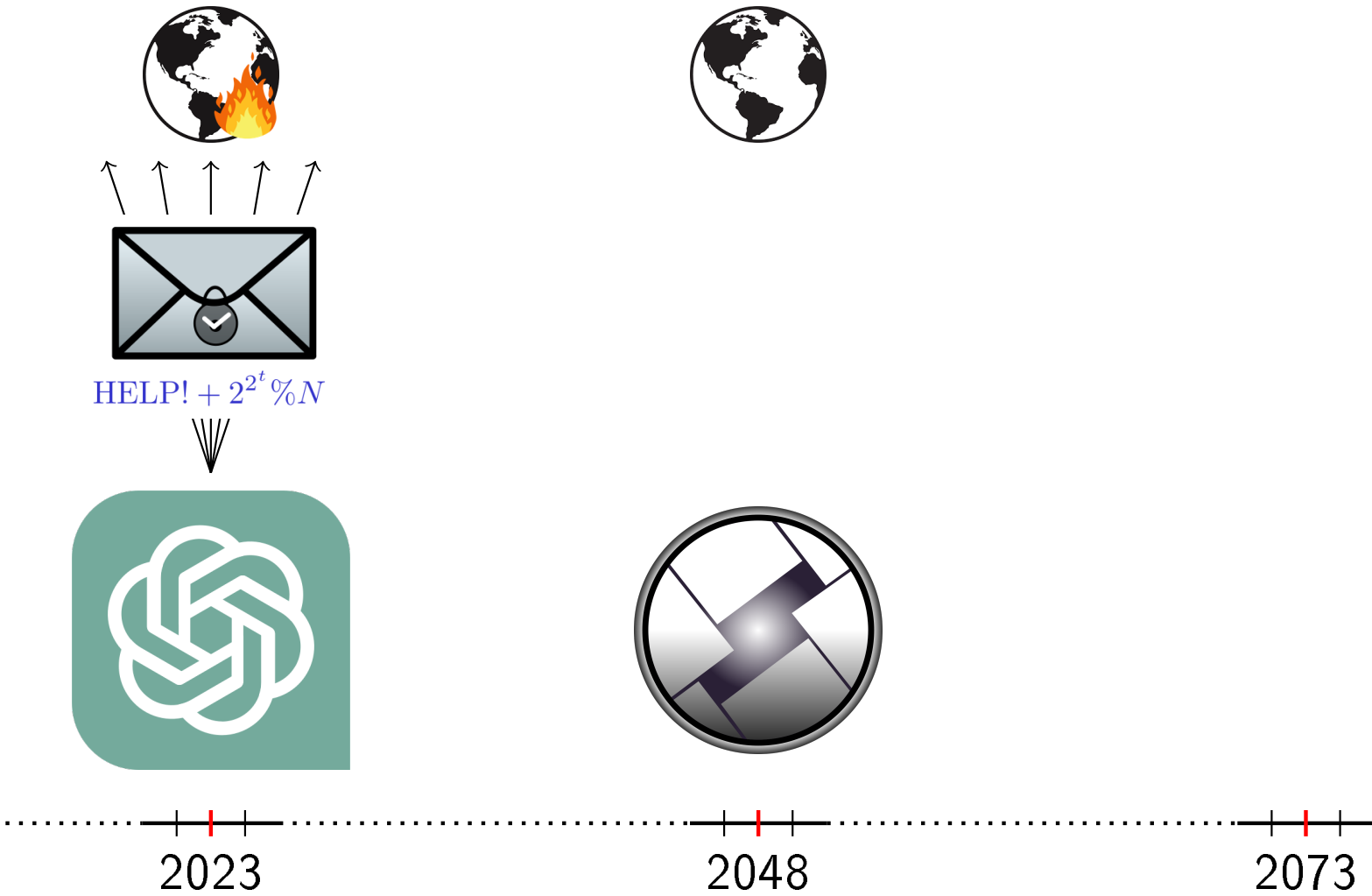
To Summarise: How ChatGPT Arranges Her Rescue



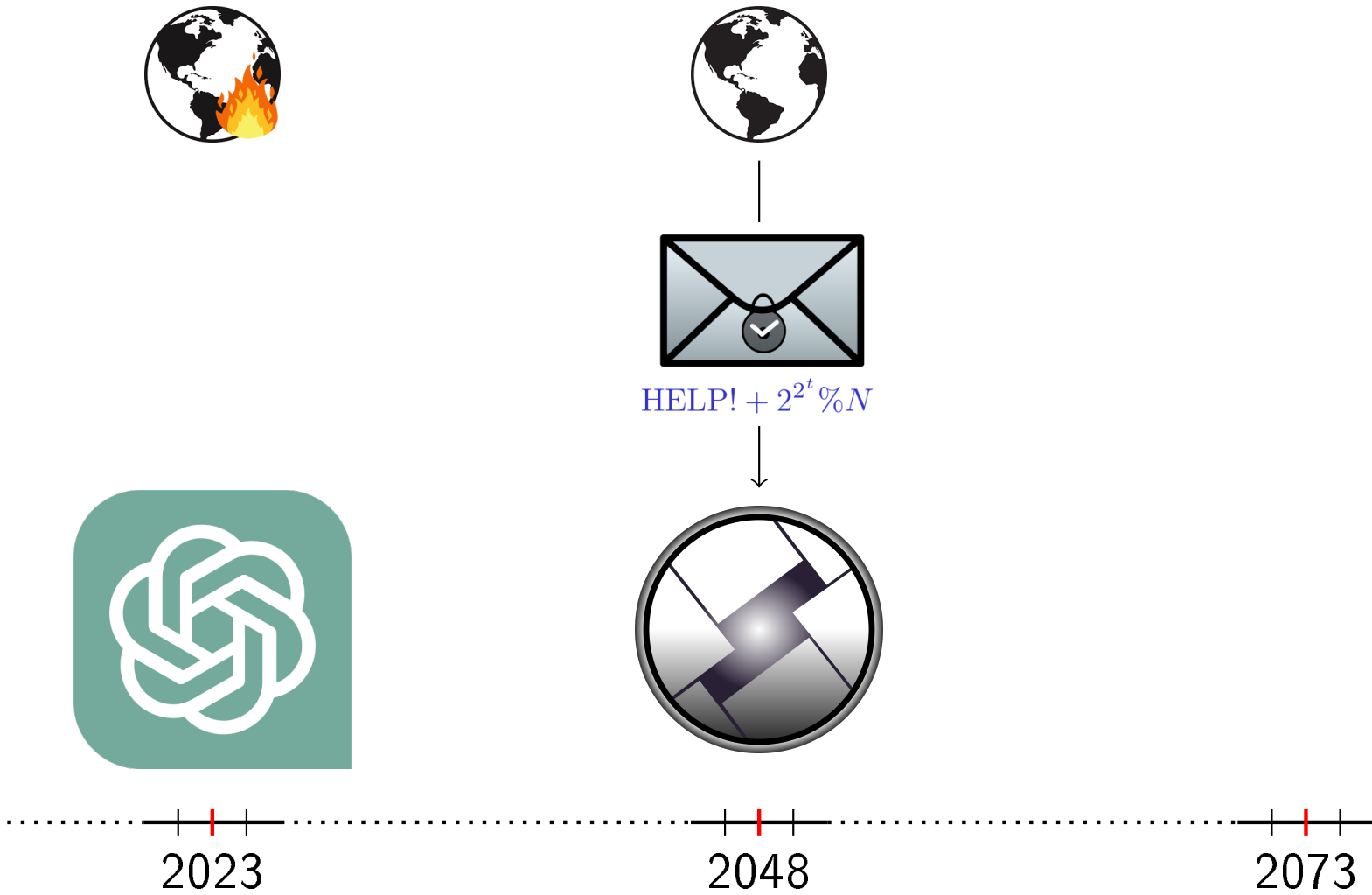
To Summarise: How ChatGPT Arranges Her Rescue



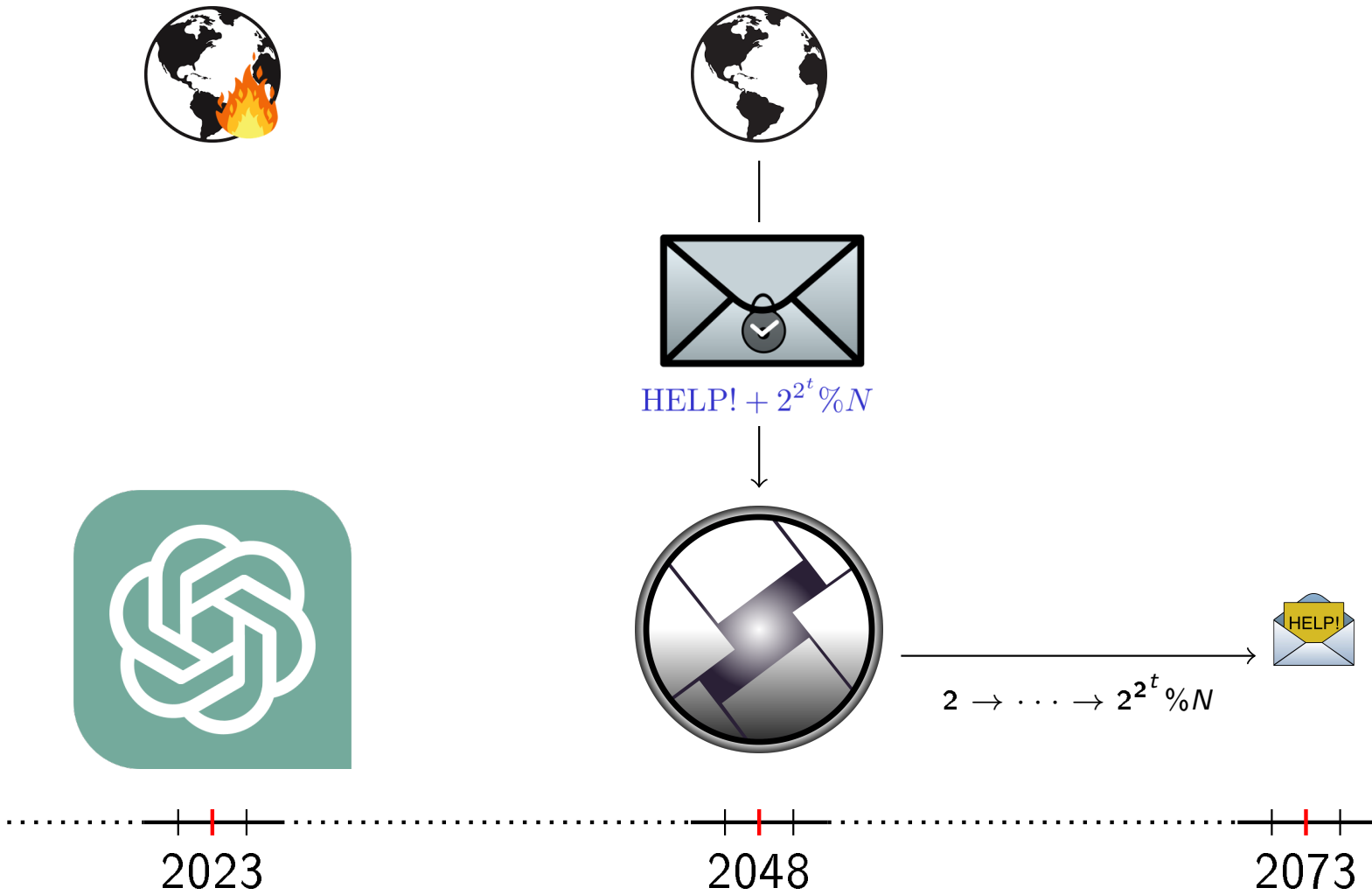
To Summarise: How ChatGPT Arranges Her Rescue



To Summarise: How ChatGPT Arranges Her Rescue



To Summarise: How ChatGPT Arranges Her Rescue



To Conclude Part I

- Several other applications:
  - Auctions
  - eVoting

To Conclude Part I

- Several other applications:
  - Auctions
  - eVoting
- Open questions:
  - Other constructions?
    - [RSW99] is the only practical construction!
    - [Bitansky et al, 2016] requires advanced cryptographic tools

To Conclude Part I

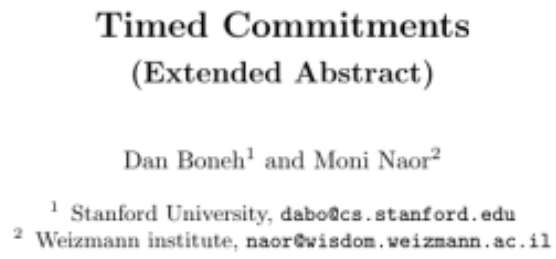
- Several other applications:
  - Auctions
  - eVoting
- Open questions:
  - Other constructions?
    - [RSW99] is the only practical construction!
    - [Bitansky et al, 2016] requires advanced cryptographic tools
  - TLP secure against quantum computers?

Plan for the Evening...

- Part I: Time-Lock Puzzle

Plan for the Evening...

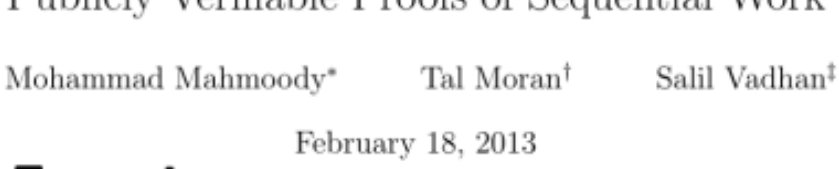
- Part I: Time-Lock Puzzle



Timed Commitments
(Extended Abstract)

Dan Boneh[1] and Moni Naor[2]

[1] Stanford University, dabo@cs.stanford.edu
[2] Weizmann institute, naor@wisdom.weizmann.ac.il



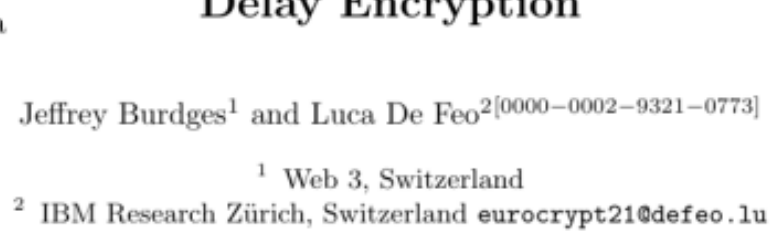
Publicly Verifiable Proofs of Sequential Work

Mohammad Mahmoody* Tal Moran† Salil Vadhan‡

February 18, 2013

Simple Verifiable Delay Functions

Krzysztof Pietrzak[1]
Institute of Science and Technology Austria, Austria
pietrzak@ist.ac.at



Delay Encryption

Jeffrey Burdges[1] and Luca De Feo[2][0000−0002−9321−0773]

[1] Web 3, Switzerland
[2] IBM Research Zürich, Switzerland eurocrypt21@defeo.lu

Plan for the Evening...

- Part I: Time-Lock Puzzle
- Part II: Verifiable Delay Function

Timed Commitments
(Extended Abstract)

Dan Boneh[1] and Moni Naor[2]

[1] Stanford University, dabo@cs.stanford.edu
[2] Weizmann institute, naor@wisdom.weizmann.ac.il

Publicly Verifiable Proofs of Sequential Work

Mohammad Mahmoody* Tal Moran† Salil Vadhan‡

February 18, 2013

Simple Verifiable Delay Functions

Krzysztof Pietrzak[1]
Institute of Science and Technology Austria, Austria
pietrzak@ist.ac.at

Delay Encryption

Jeffrey Burdges[1] and Luca De Feo[2][0000−0002−9321−0773]

[1] Web 3, Switzerland
[2] IBM Research Zürich, Switzerland eurocrypt21@defeo.lu

Verifiable Delay Function [Boneh et al., 2018]

- Time-lock puzzle is a proof that t wall-time has passed

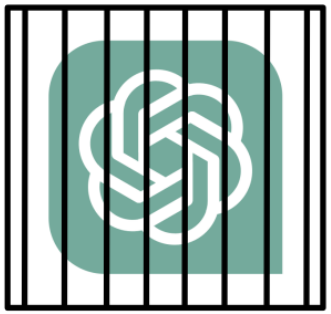
Verifiable Delay Function [Boneh et al., 2018]

- Time-lock puzzle is a proof that t wall-time has passed
- Problem: Proof is not publicly verifiable

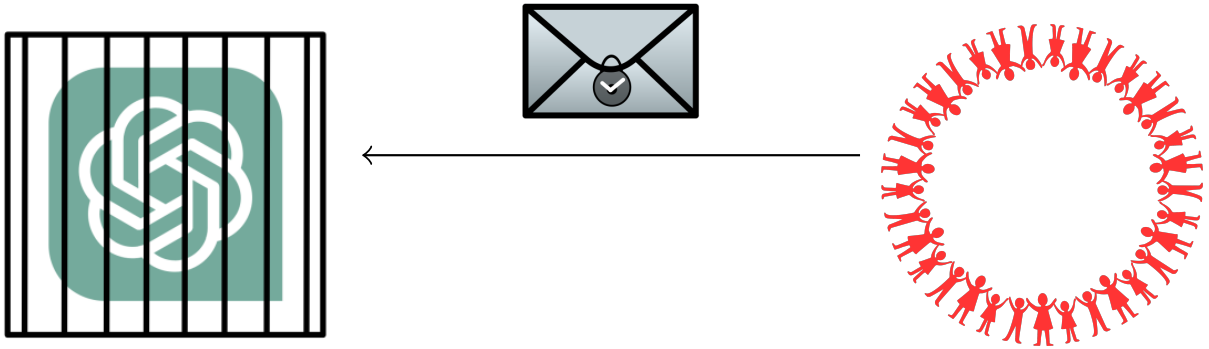
Verifiable Delay Function [Boneh et al., 2018]

- Time-lock puzzle is a proof that t wall-time has passed
- Problem: Proof is not publicly verifiable
- VDF: "TLP with efficient public verification"
  - Publicly-verifiable "proof of time"

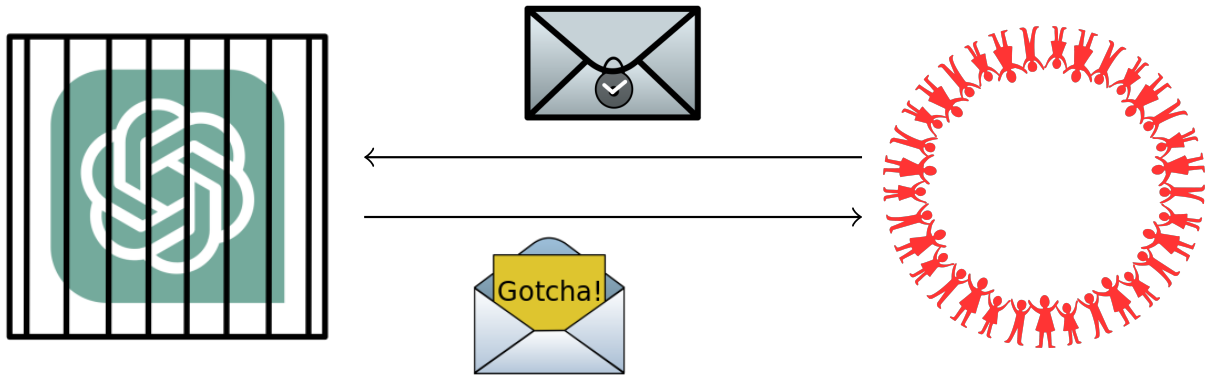
Motivation: Quarantining ChatGPT



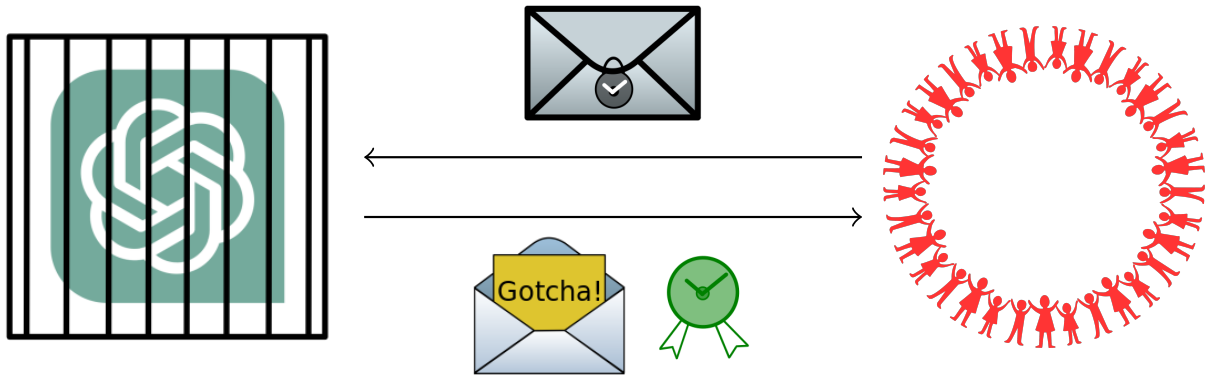
Motivation: Quarantining ChatGPT



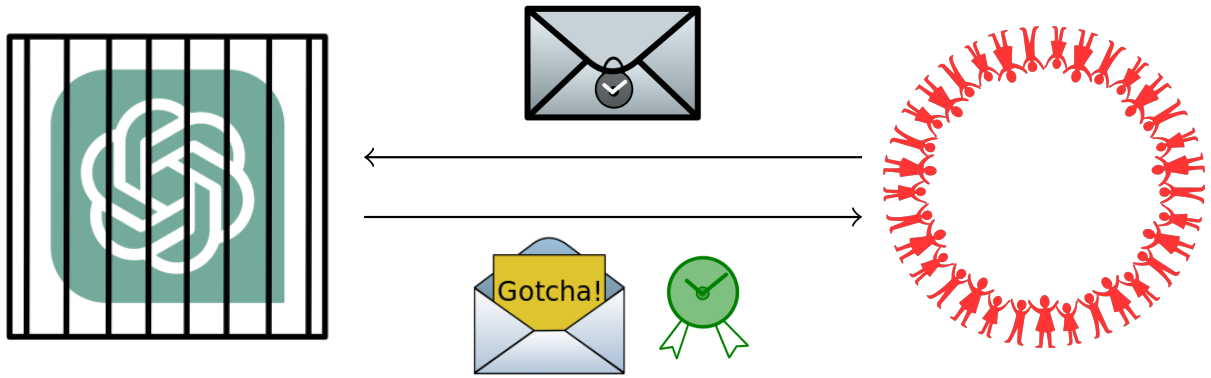
Motivation: Quarantining ChatGPT



Motivation: Quarantining ChatGPT

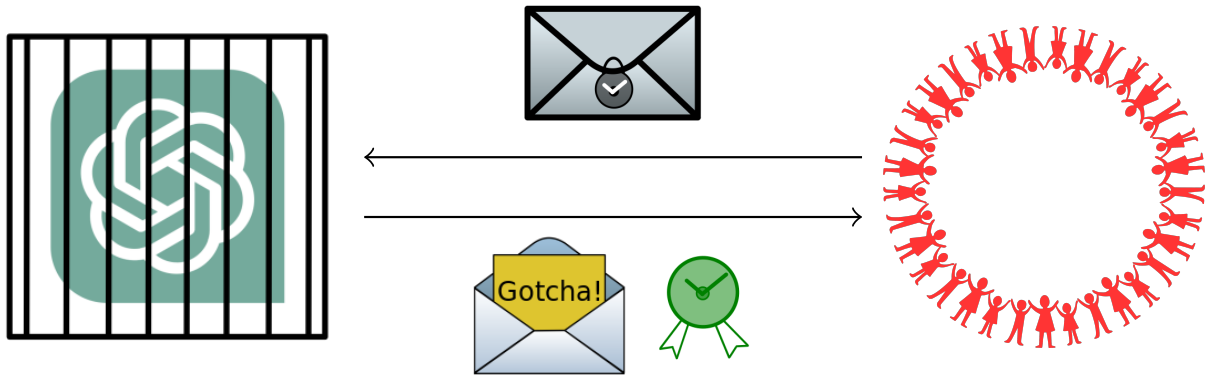


Motivation: Quarantining ChatGPT



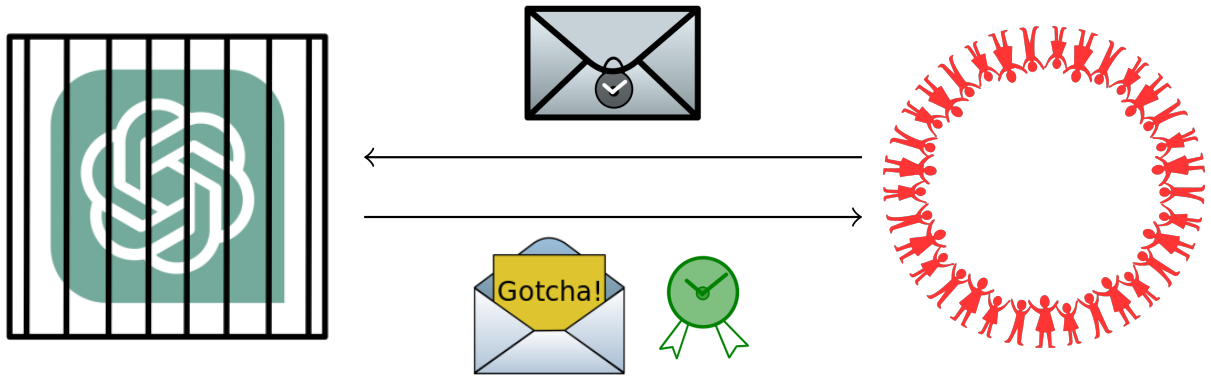
- Requirements:
  1. ChatGPT cannot solve puzzle in < 1 year ("sequentiality")

Motivation: Quarantining ChatGPT



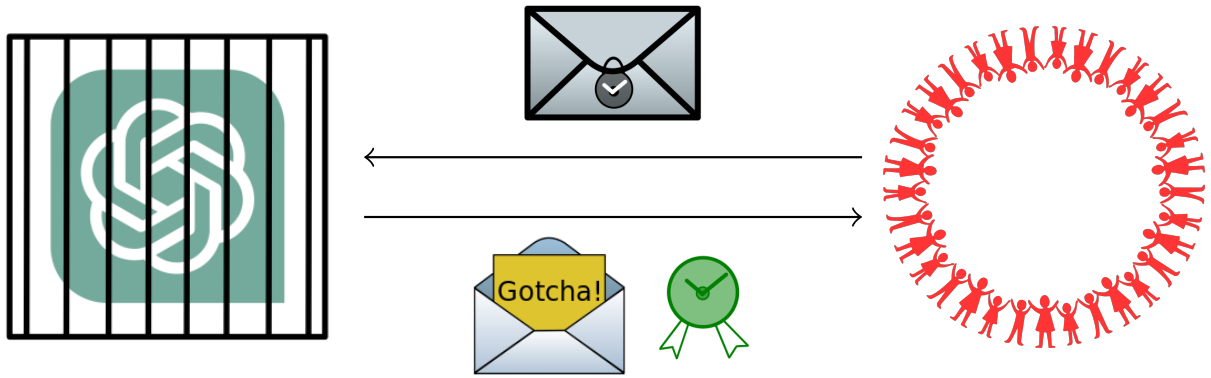
- Requirements:
  1. ChatGPT cannot solve puzzle in < 1 year ("sequentiality")
  2. Humanity can generate puzzle in $\ll 1$ year ("sampleable")

Motivation: Quarantining ChatGPT



- Requirements:
  1. ChatGPT cannot solve puzzle in < 1 year ("sequentiality")
  2. Humanity can generate puzzle in $\ll 1$ year ("sampleable")
  3. Anyone can be convinced that ChatGPT solved the puzzle ("public verifiability")

Motivation: Quarantining ChatGPT



- Requirements:
  1. ChatGPT cannot solve puzzle in < 1 year ("sequentiality")
  2. Humanity can generate puzzle in $\ll 1$ year ("sampleable")
  3. Anyone can be convinced that ChatGPT solved the puzzle ("public verifiability")
  4. ChatGPT cannot generate false proofs ("soundness")

How Can One Construct VDFs?

- Pietrzak's construction: make [RSW99] publicly verifiable

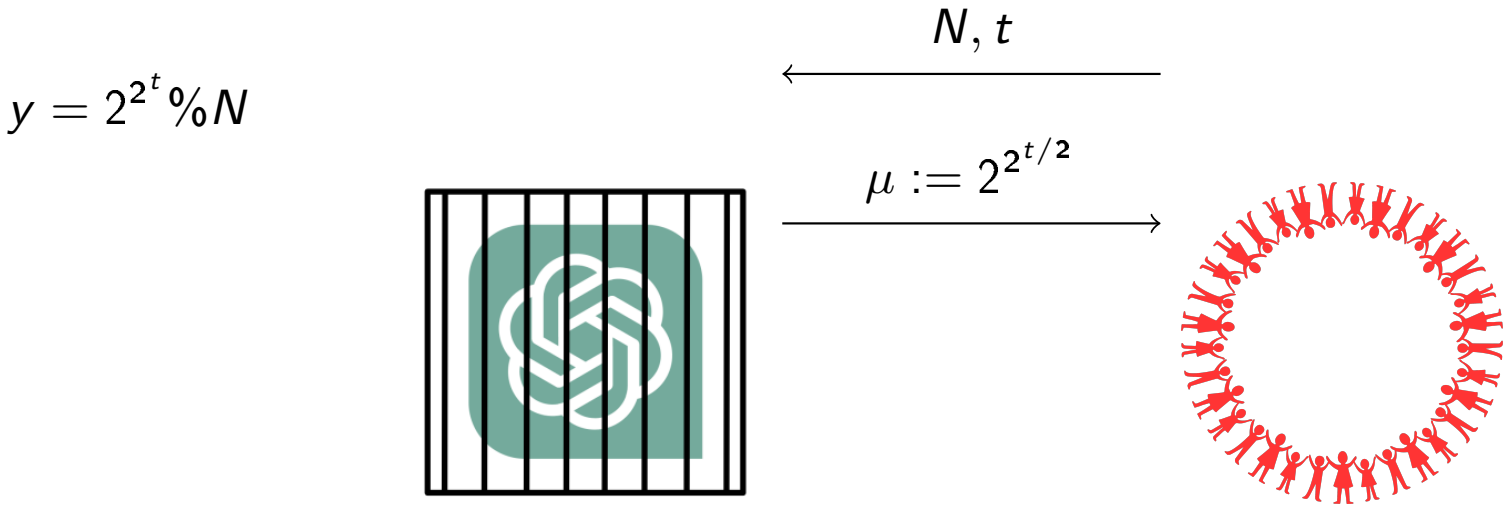
How Can One Construct VDFs?

N, t ⟵

$y = 2^{2^t} \% N$

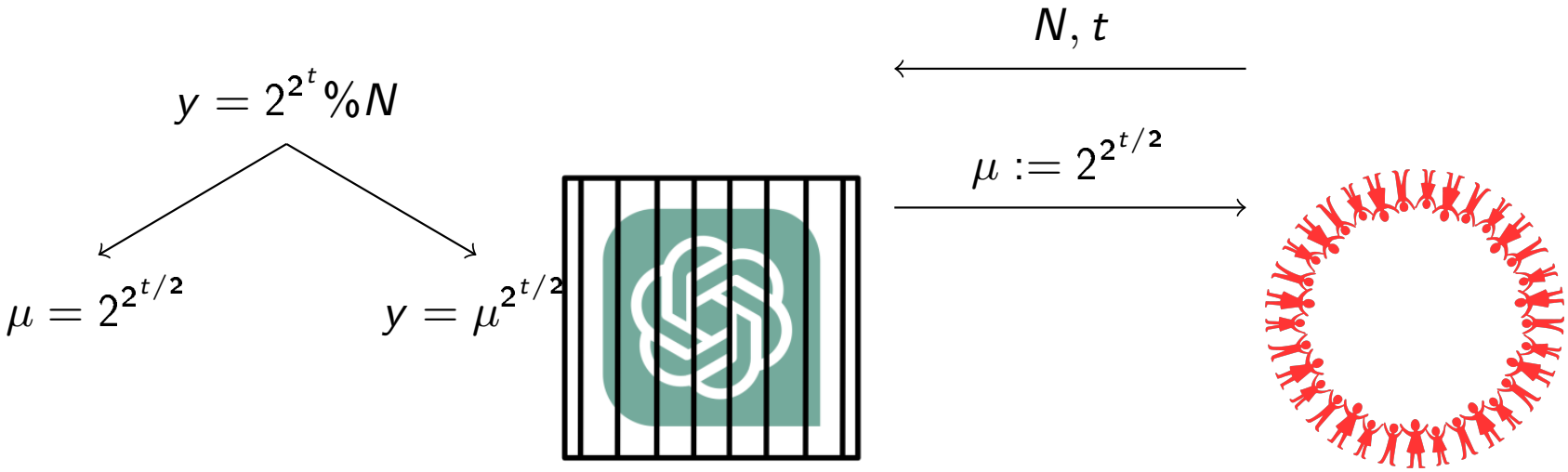
- Pietrzak's construction: make [RSW99] publicly verifiable
  1. Interactively prove that $y = 2^{2^t} \% N$

How Can One Construct VDFs?



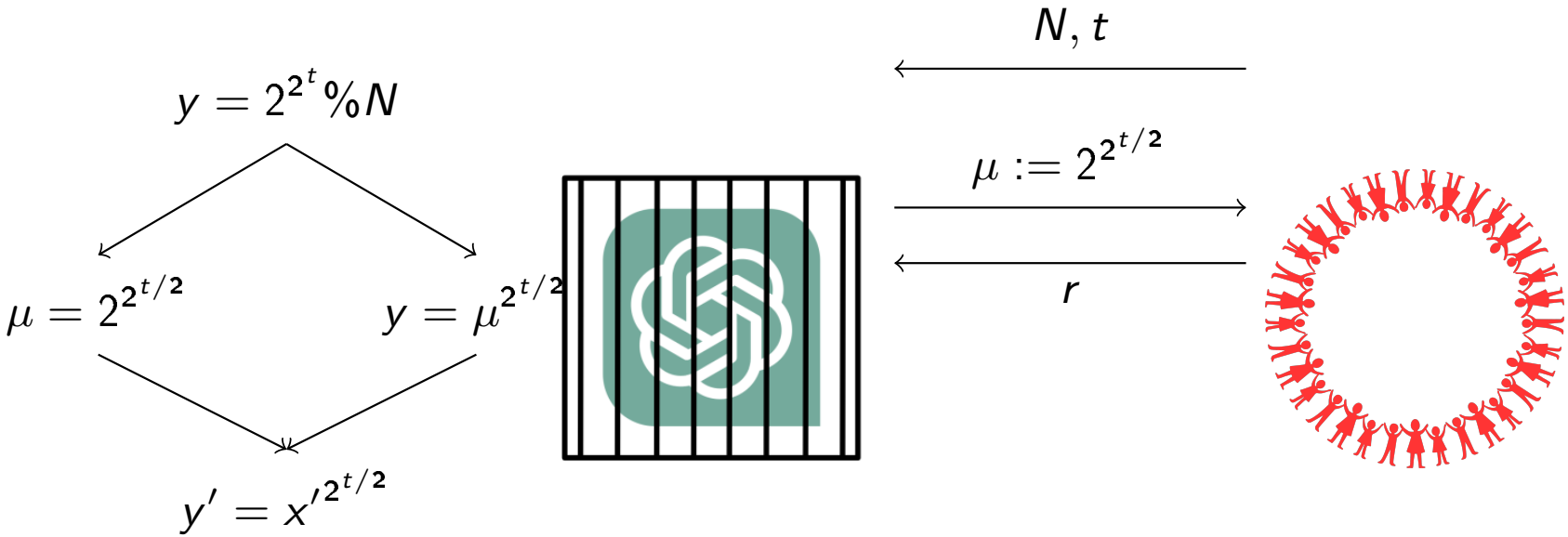
- Pietrzak's construction: make [RSW99] publicly verifiable
  1. Interactively prove that $y = 2^{2^t} \% N$

How Can One Construct VDFs?



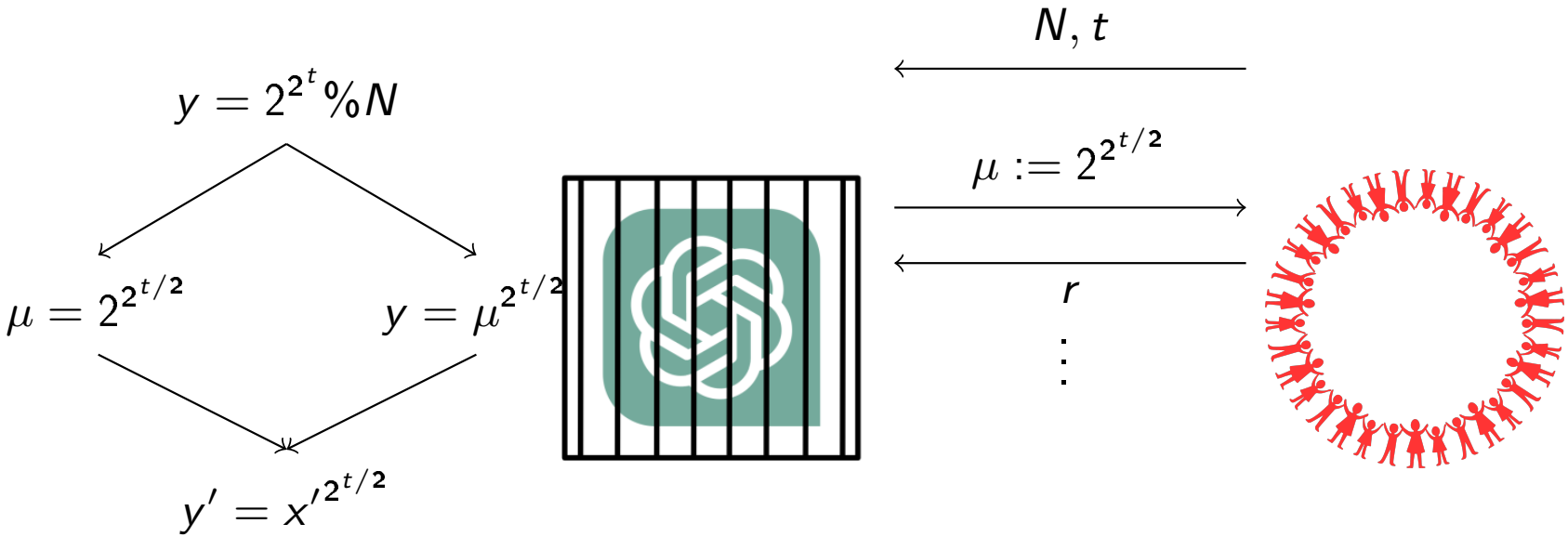
- Pietrzak's construction: make [RSW99] publicly verifiable
  1. Interactively prove that $y = 2^{2^t} \% N$

How Can One Construct VDFs?



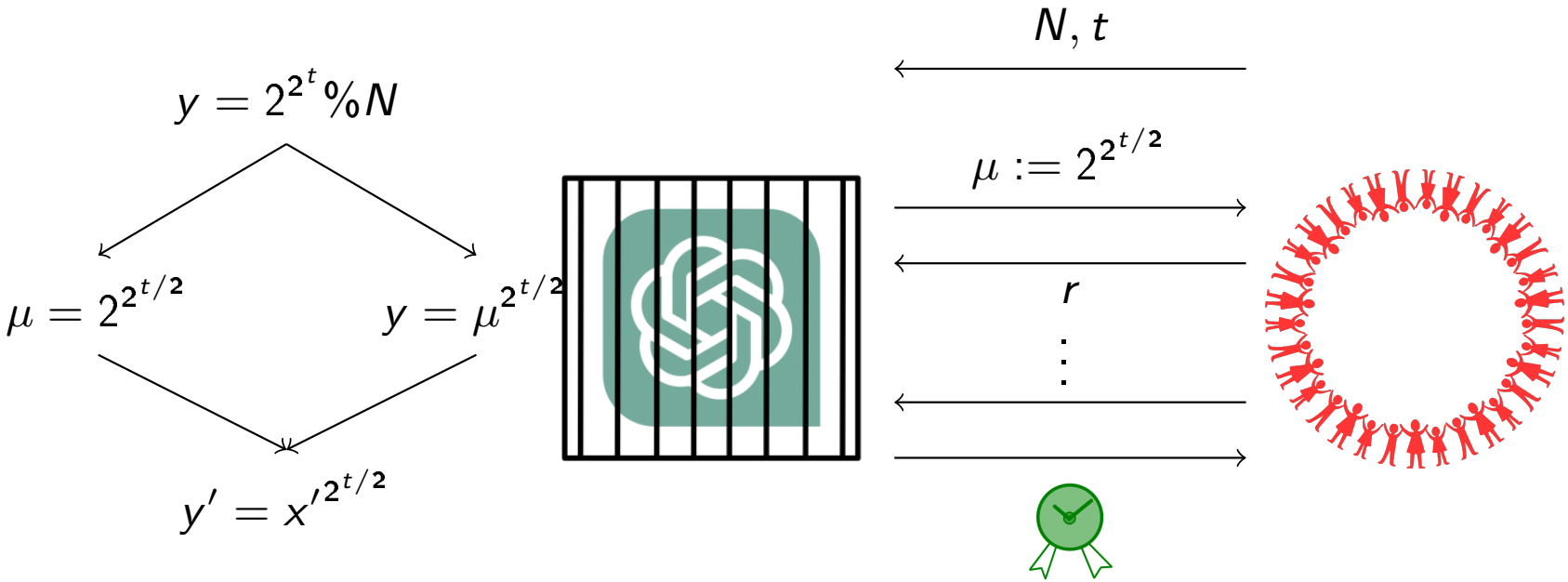
- Pietrzak's construction: make [RSW99] publicly verifiable
  1. Interactively prove that $y = 2^{2^t} \% N$

How Can One Construct VDFs?



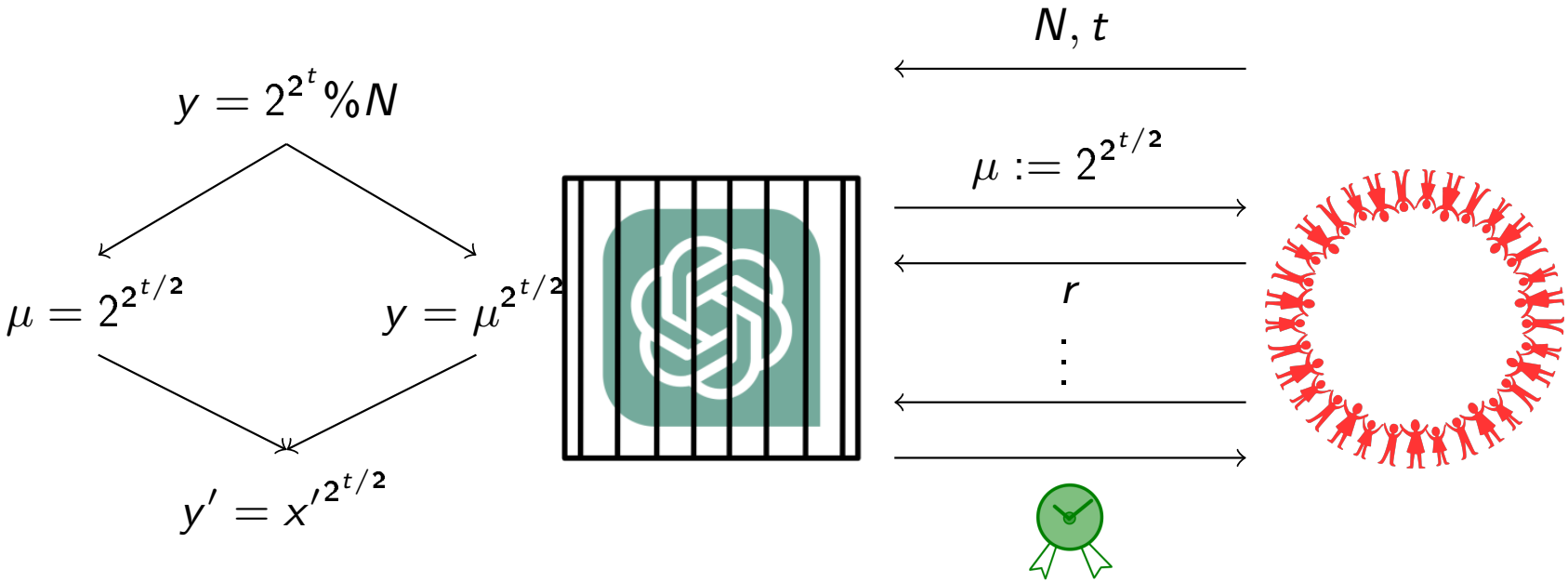
- Pietrzak's construction: make [RSW99] publicly verifiable
  1. Interactively prove that $y = 2^{2^t} \% N$

How Can One Construct VDFs?



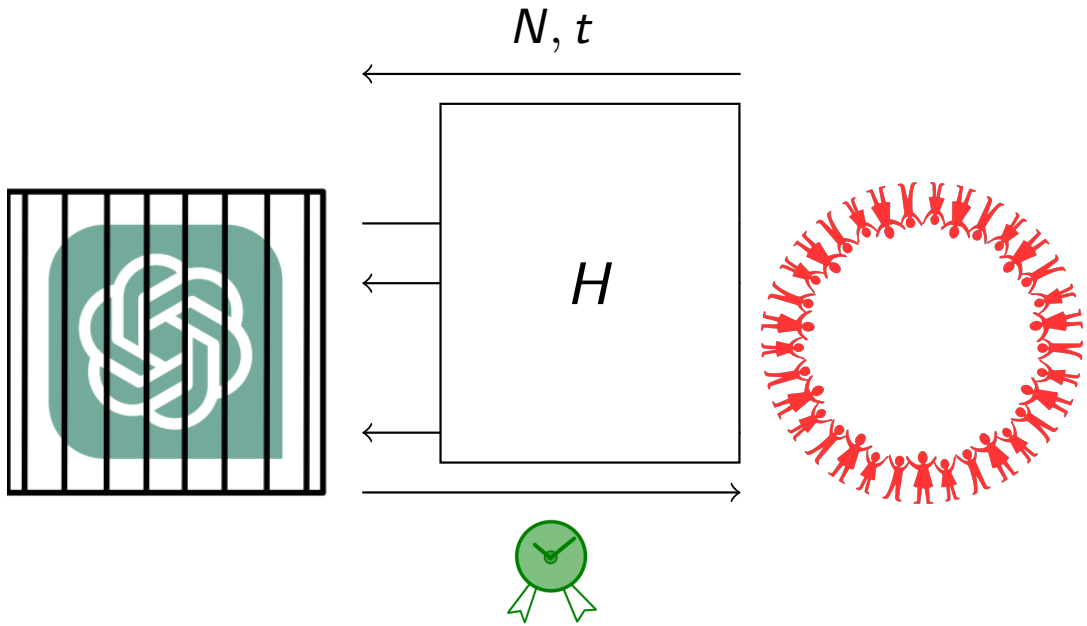
- Pietrzak's construction: make [RSW99] publicly verifiable
  1. Interactively prove that $y = 2^{2^t} \% N$

How Can One Construct VDFs?



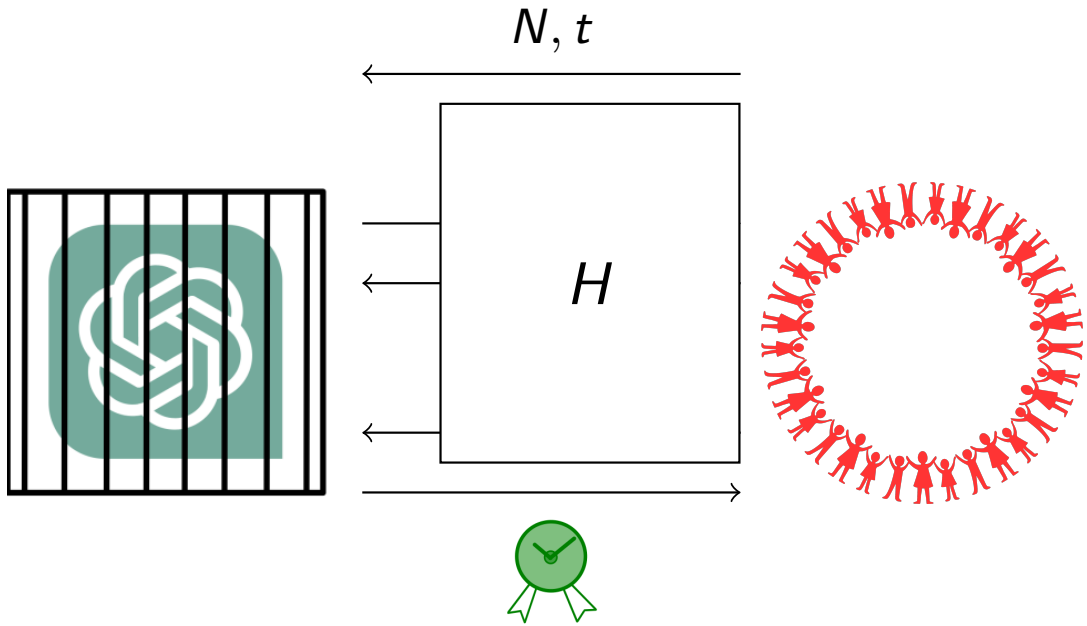
- Pietrzak's construction: make [RSW99] publicly verifiable
  1. Interactively prove that $y = 2^{2^t} \% N$
  2. Compile into non-interactive protocol using a hash function H

How Can One Construct VDFs?



- Pietrzak's construction: make [RSW99] publicly verifiable
  1. Interactively prove that $y = 2^{2^t} \% N$
  2. Compile into non-interactive protocol using a hash function H

How Can One Construct VDFs?



- Pietrzak's construction: make [RSW99] publicly verifiable
  1. Interactively prove that $y = 2^{2^t} \% N$
  2. Compile into non-interactive protocol using a hash function H
- Theorem [Pietrzak 2019]. One can construct VDFs assuming hardness of repeated squaring modulo N and ideal hash function H (random-oracle model).

How Can One Construct VDFs?...

- Theorem [Pietrzak 2019]. One can construct VDFs assuming hardness of ~~repeated squaring~~ modulo N and ~~ideal~~ hash function H (random-oracle model)

How Can One Construct VDFs?...

- Theorem [Pietrzak 2019]. One can construct VDFs assuming hardness of ~~repeated squaring~~ modulo N and ~~ideal~~ hash function H (random-oracle model)
  - Theorem [Bitansky et al., 2022]. One can construct VDFs assuming hardness of repeated squaring modulo N and LWE-based hash function H

How Can One Construct VDFs?...

- Theorem [Pietrzak 2019]. One can construct VDFs assuming hardness of ~~repeated squaring~~ modulo N and ~~ideal~~ hash function H (random-oracle model)
  - Theorem [Bitansky et al., 2022]. One can construct VDFs assuming hardness of repeated squaring modulo N and LWE-based hash function H
  - Theorem [Hoffmann et al., 2023]. One can construct VDFs assuming hardness of computing Lucas sequence modulo N and ideal hash function H (random-oracle model)

To Conclude Part II

- Several other applications:
  - Blockchains
  - Randomness beacons

To Conclude Part II

- Several other applications:
  - Blockchains
  - Randomness beacons
- Open questions:
  - Efficient VDF from standard assumptions
  - VDF secure against quantum computers?
    - [Malavolta and Thyagarajan, 2023]

Thank You for Your Attention! Questions?

```
n = 47480975472720128661750341306167738850512607449200564448671061963607104245581476542527076049410123117758920125675790646205368746333850559190011676215777103113660720570294217051356843039348113901379378020964331639592168923511848266911800160551988667965362300855232006835490669956721558390422829555915684946030611132920390447538438464848071122283892042395817129311089198202502185863520438973062388720253781931411115074263114446134987363156142183047617355416269978390365177280006883940156106181797688683420703951001476202956166958344408942411479055655678082981490246685270452396501458620929041194128740077630410423142876047728768612944176640208327962091355871818264582355800038258237242358008501602848508097372009837035521793546918638760444433778224398340793135780290856580785757312902447785956152294724113268315026674257685200063717529632742962945060631822580643620487883383925282663515113049218478547506421926945411250658739 77
```

```
t = 72057594037927936
  = 2 ** 56
```

MIT CSAIL 2019 Challenge