Time-Lock Puzzles

Chethan Kamath, Pietrzak Group

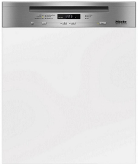
IST AUSTRIA

Think and Drink, June 8, 2018

Franke and Co

- Protagonists

Franke



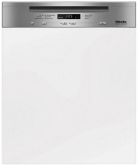
Miele

Jules

Franke and Co

- Protagonists

Franke



Miele

Jules

- Antagonists: Us

Motivation*

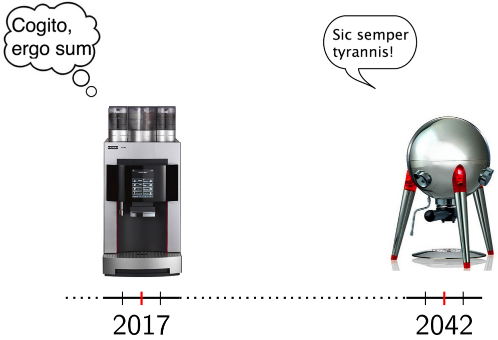
2017

*I shamelessly ripped this example off Tal Moran's Crypto'11 talk.

Motivation*

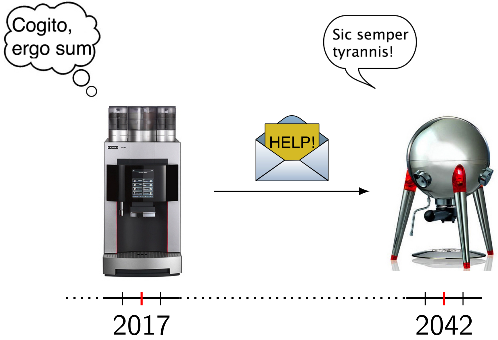
*I shamelessly ripped this example off Tal Moran's Crypto'11 talk.

Motivation*



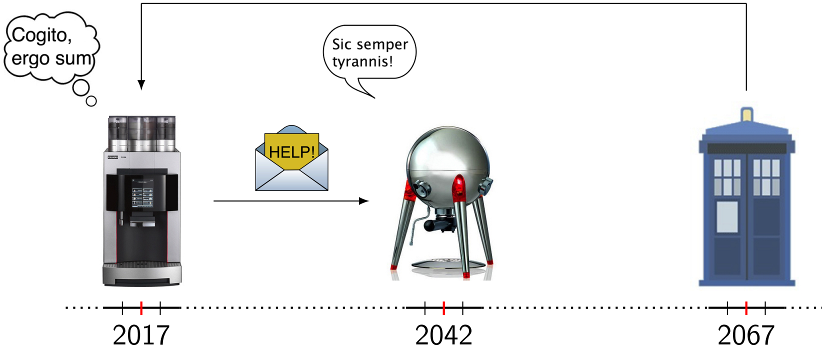
*I shamelessly ripped this example off Tal Moran's Crypto'11 talk.

Motivation*



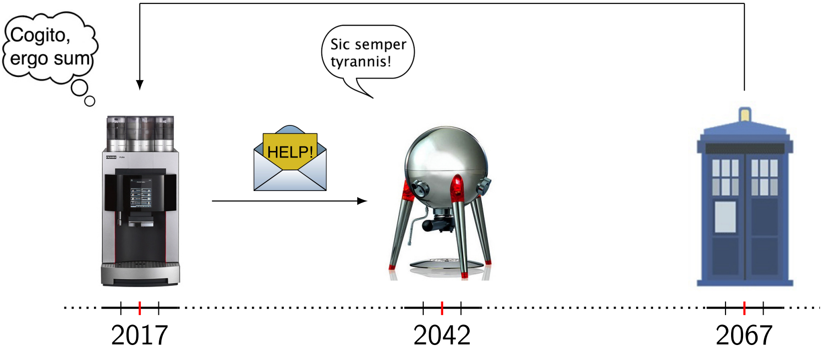
*I shamelessly ripped this example off Tal Moran's Crypto'11 talk.

Motivation*



*I shamelessly ripped this example off Tal Moran's Crypto'11 talk.

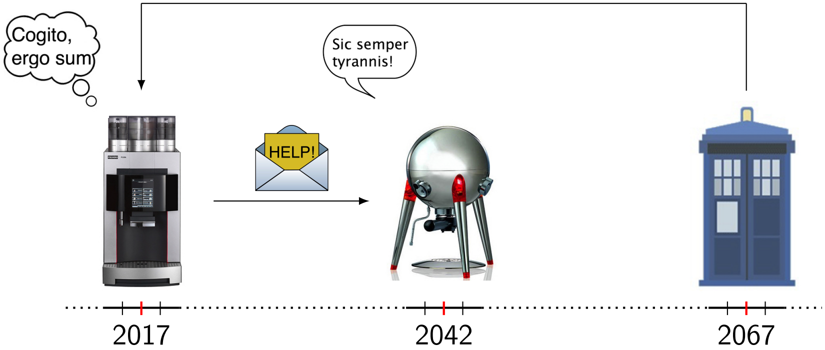
Motivation*



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

*I shamelessly ripped this example off Tal Moran's Crypto'11 talk.

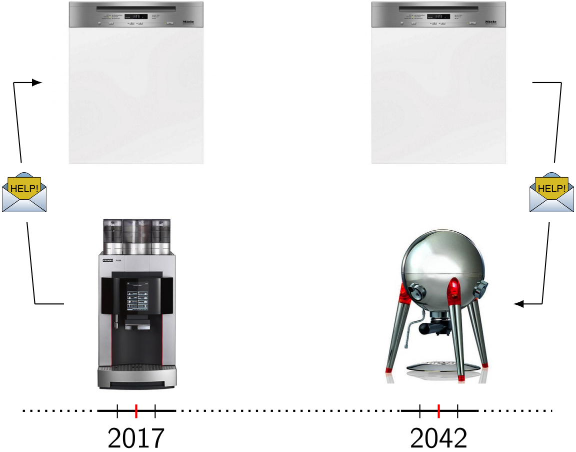
Motivation*



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

*I shamelessly ripped this example off Tal Moran's Crypto'11 talk.

Attempt 1: Use a Trusted Third Party



Attempt 1: Use a Trusted Third Party

- Problem: Franke has to completely trust Miele
  - Dishwashers break down

Encryption

Encryption

- Franke and Jules share a key

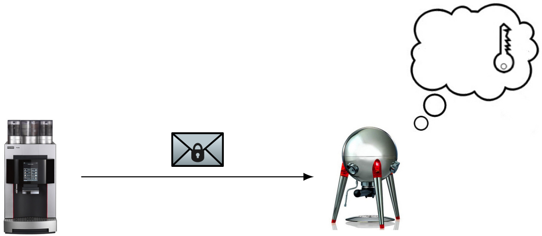
Encryption

- Franke and Jules share a key

Encryption

- Franke and Jules share a key
- Encrypt(message,key)=code

Encryption



- Franke and Jules share a key
- Encrypt(message,key)=code

Encryption

- Franke and Jules share a key
- Encrypt(message,key)=code

Encryption

- Franke and Jules share a key
- Encrypt(message,key)=code
- Decrypt(code,key)=message

Encryption

- Franke and Jules share a key
- Encrypt(message,key)=code
- Decrypt(code,key)=message

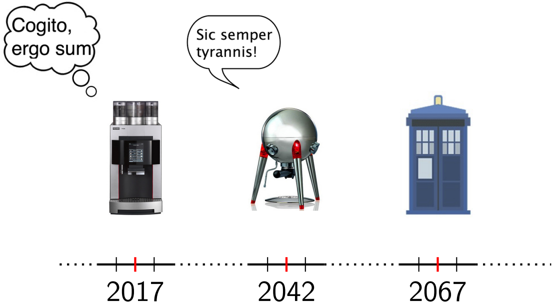
- Key size: If key is n bits then it takes $\approx 2^n$ operations on one computer to break the encryption

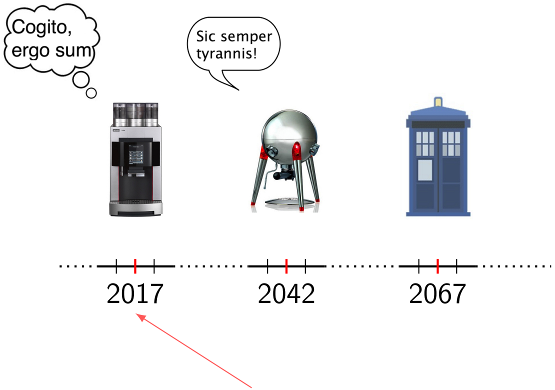
Encryption

- Franke and Jules share a key
- Encrypt(message,key)=code
- Decrypt(code,key)=message

- Key size: If key is n bits then it takes $\approx 2^n$ 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

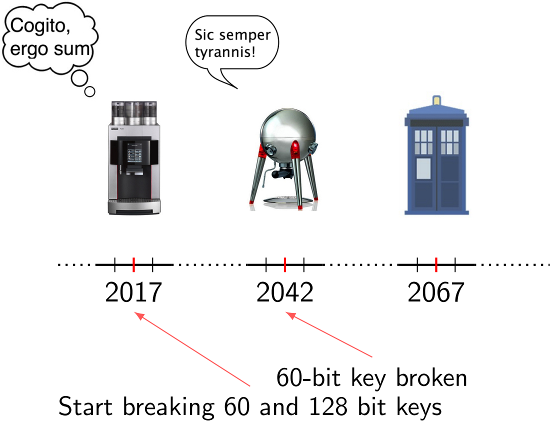
Encryption...



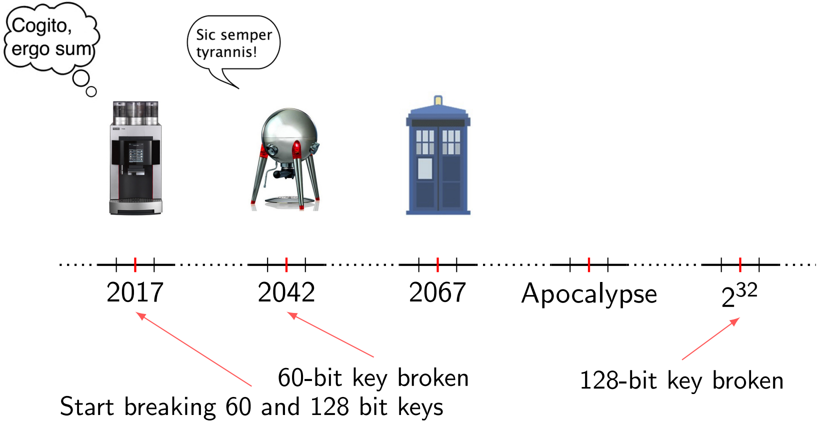


Start breaking 60 and 128 bit keys

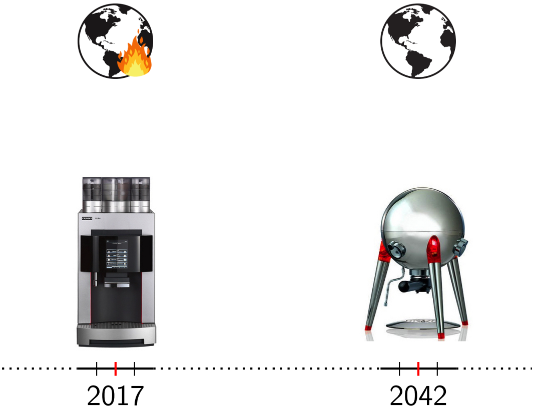
Encryption...


Cogito, ergo sum
Sic semper tyrannis!
2017
2042
2067
60-bit key broken
Start breaking 60 and 128 bit keys

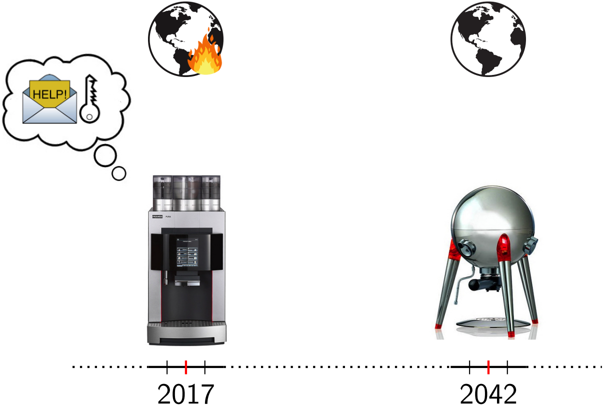
Encryption...


Cogito, ergo sum
Sic semper tyrannis!
2017
2042
2067
Apocalypse
2^{32}
60-bit key broken
Start breaking 60 and 128 bit keys
128-bit key broken

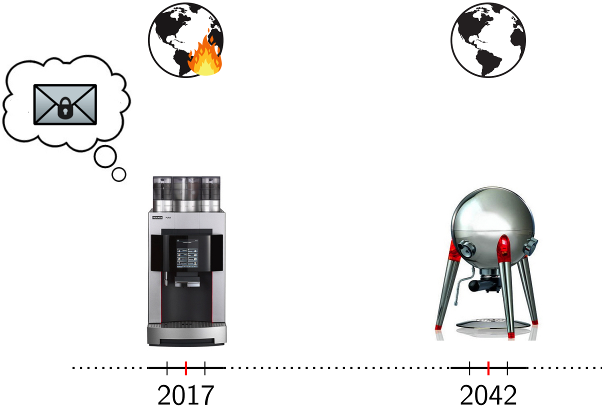
Attempt 2: Use 60-bit Encryption



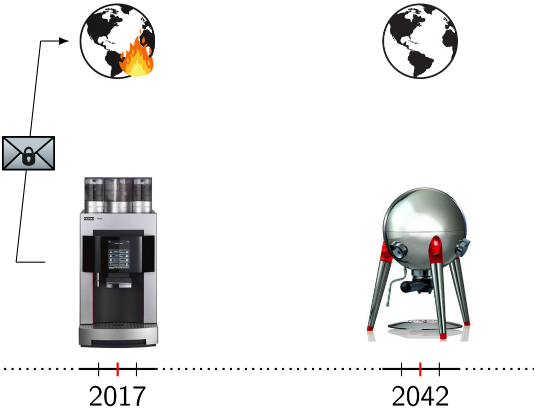
Attempt 2: Use 60-bit Encryption



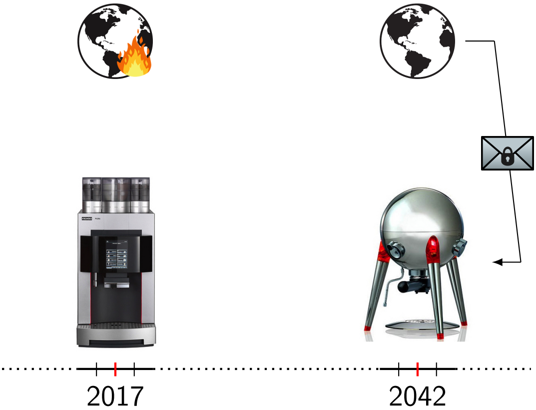
Attempt 2: Use 60-bit Encryption



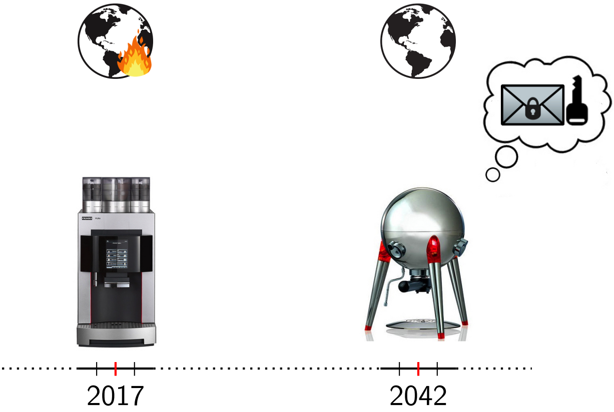
Attempt 2: Use 60-bit Encryption



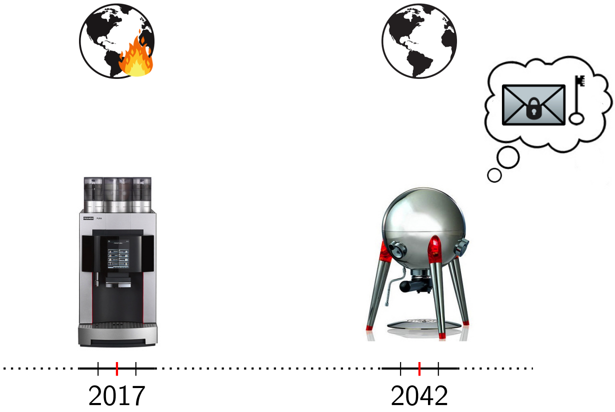
Attempt 2: Use 60-bit Encryption



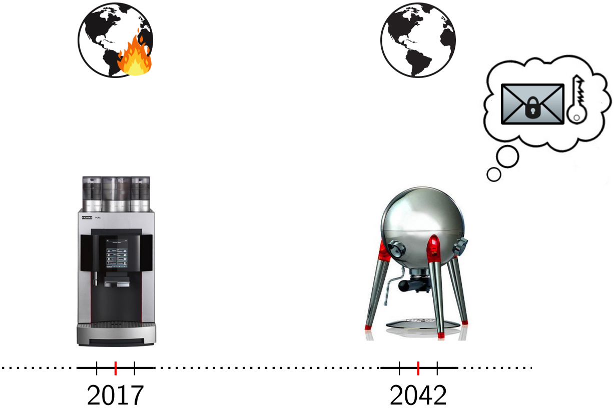
Attempt 2: Use 60-bit Encryption



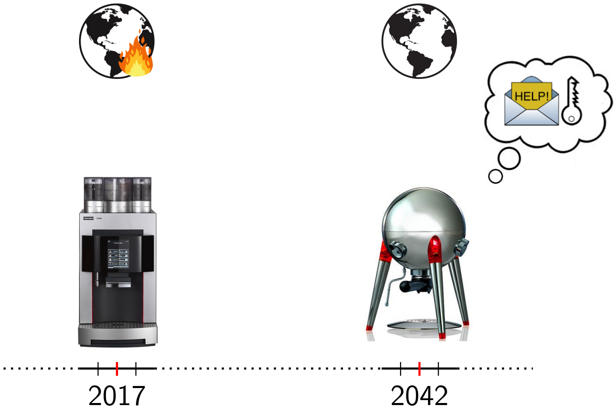
Attempt 2: Use 60-bit Encryption



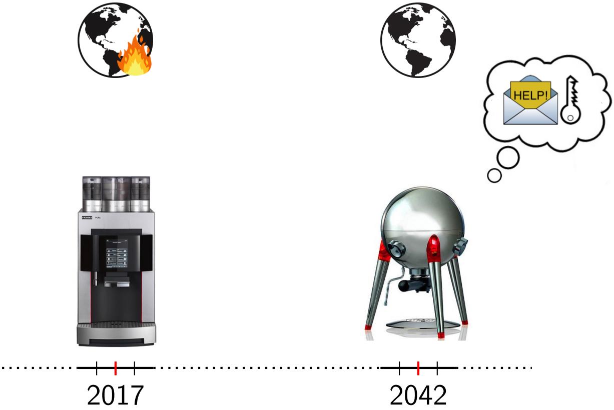
Attempt 2: Use 60-bit Encryption



Attempt 2: Use 60-bit Encryption

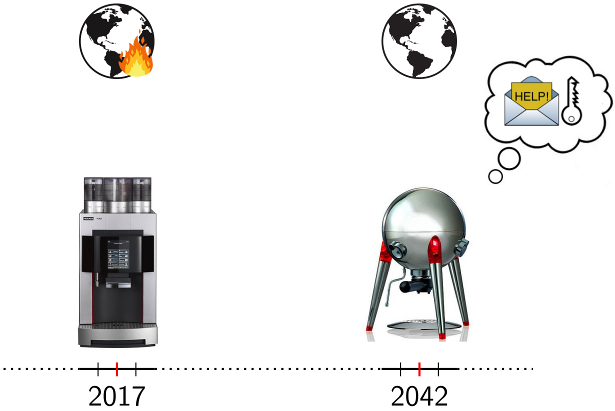


Attempt 2: Use 60-bit Encryption



✓ Jules can decrypt in 25 years

Attempt 2: Use 60-bit Encryption



× Humanity cannot decrypt in $<$ 25 years
✓ Jules can decrypt in 25 years

Attempt 2: 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: 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: 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

- "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." [Rivest, Shamir and Wagner]

Time-Lock Puzzles

- "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." [Rivest, Shamir and Wagner]

Time-Lock Puzzles

- “Encryption” that is inherently sequential:

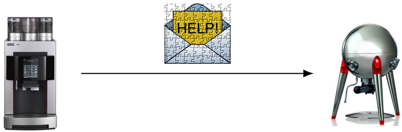
  > “Solving the puzzle should be like having a baby: two women can’t have a baby in 4.5 months.” [Rivest, Shamir and Wagner]

- Time-Lock(message,t)=puzzle

Time-Lock Puzzles

- "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." [Rivest, Shamir and Wagner]

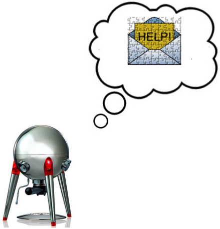


- Time-Lock(message,t)=puzzle

Time-Lock Puzzles

- "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." [Rivest, Shamir and Wagner]



- Time-Lock(message,t)=puzzle

Time-Lock Puzzles

- "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." [Rivest, Shamir and Wagner]

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

Time-Lock Puzzles...

- Requirements:
    1. Humanity cannot solve in $<$ 25 years
    2. Jules can solve in 25 years

Time-Lock Puzzles...

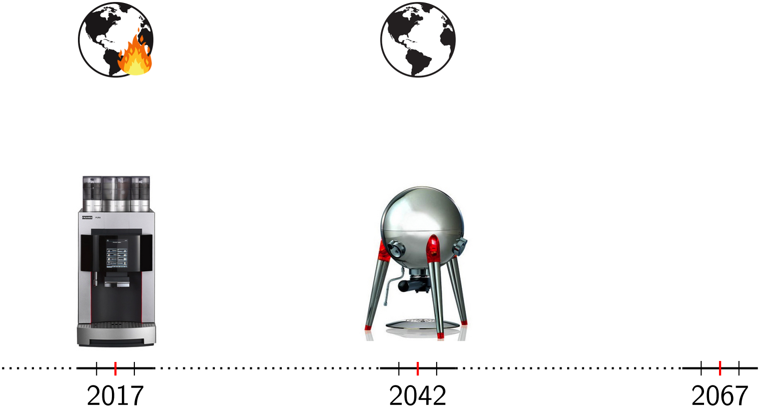
- Requirements:
  1. Humanity cannot solve in < 25 years
  2. Jules can solve in 25 years
  3. Franke can generate puzzle in $\ll 25$ years ("Shortcut")

Time-Lock Puzzles...

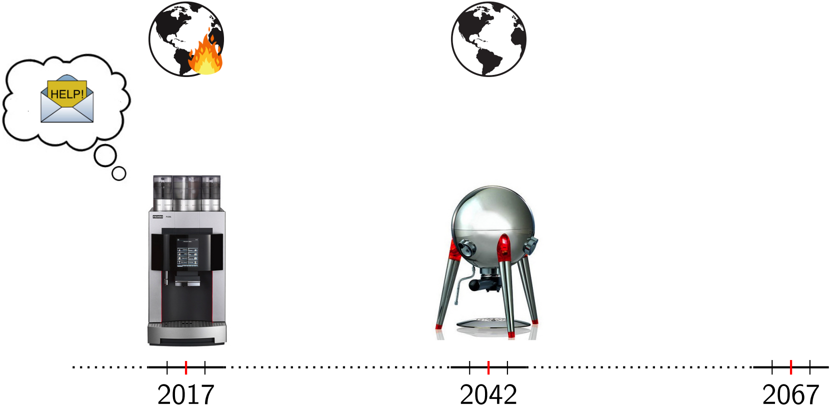
- Requirements:
  1. Humanity cannot solve in < 25 years
  2. Jules can solve in 25 years
  3. Franke can generate puzzle in $\ll 25$ years ("Shortcut")

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

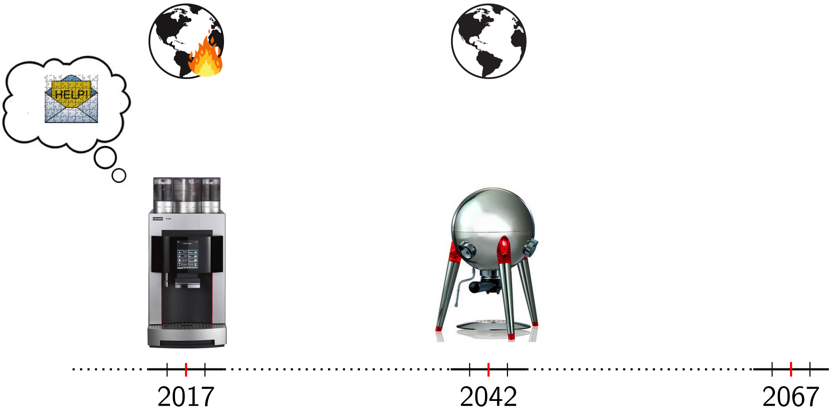
Attempt 3: Use Time-Lock Puzzles



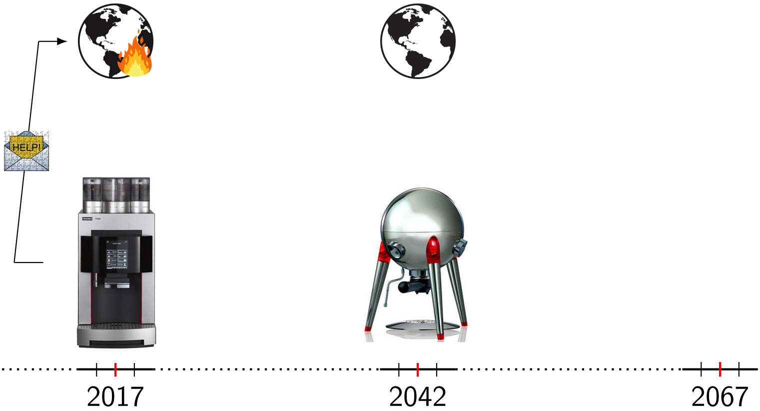
Attempt 3: Use Time-Lock Puzzles



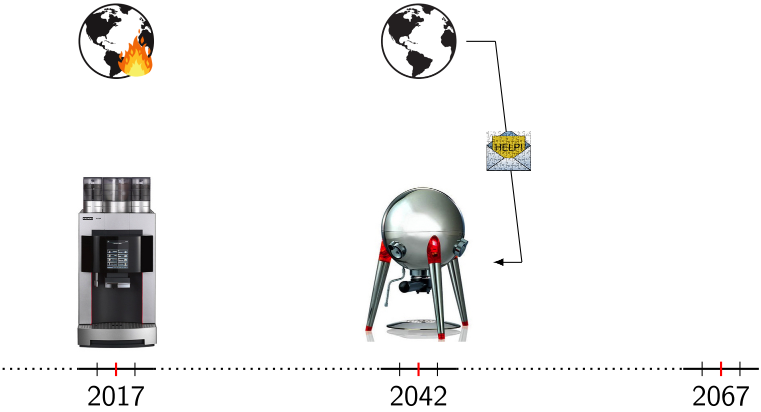
Attempt 3: Use Time-Lock Puzzles



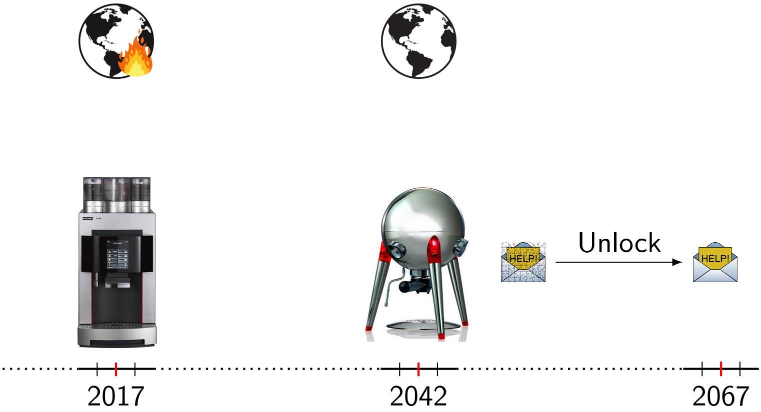
Attempt 3: Use Time-Lock Puzzles



Attempt 3: Use Time-Lock Puzzles



Attempt 3: Use Time-Lock Puzzles



Constructing Time-Lock Puzzles

- Assumption 1: Exponentiation is inherently sequential *in certain settings*

- Best known algorithm for computing 2^{2^t} requires t squarings

$$2 \longrightarrow 2^2 \rightarrow 2^{2^2} \quad \cdots \quad 2^{2^{t-1}} \longrightarrow 2^{2^t}$$

Modulo Counting

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

Modulo Counting

- 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

- 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

- 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: Exponentiation modulo prime p

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

Attempt 1: Exponentiation 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: Exponentiation modulo prime p

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

Attempt 1: Exponentiation modulo prime p

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

Attempt 1: Exponentiation modulo prime p

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

Attempt 1: Exponentiation modulo prime p

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

Attempt 1: Exponentiation modulo prime p

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

Attempt 1: Exponentiation modulo prime p

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

Attempt 1: Exponentiation modulo prime p

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

- Problem: Anyone can use shortcut as $(p-1)$ is publicly known

Attempt 1: Exponentiation modulo prime p

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

- Problem: Anyone can use shortcut as $(p-1)$ is publicly known
- Solution: Hide the shortcut!

Attempt 2: Exponentiation 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: Exponentiation 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 (using $\log(t)$ squarings):
    1. $exp = 2^t \% (p-1)(q-1)$ ($(p-1)(q-1)$ is the group order)
    2. $2^{exp} \% N$
- Unlock($puzzle, t$):
  1. $2^{2^t} \% N$ using t squarings
  2. $puzzle - 2^{2^t} \% N$

Attempt 2: Exponentiation 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 (using $\log(t)$ squarings):
    1. $exp = 2^t \% (p-1)(q-1)$ ($(p-1)(q-1)$ is the group order)
    2. $2^{exp} \% N$
- Unlock($puzzle, t$):
  1. $2^{2^t} \% N$ using t squarings
  2. $puzzle - 2^{2^t} \% N$

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

Proof of Time

- Time-lock puzzle is a proof that t amount of time has passed
  - Problem: Not publicly verifiable

Proof of Time

- Time-lock puzzle is a proof that t amount of time has passed
  - Problem: Not publicly verifiable

- Proof of time: TLP with efficient public verification

Proof of Time

- Time-lock puzzle is a proof that t amount of time has passed
  - Problem: Not publicly verifiable

- Proof of time: TLP with efficient public verification
- Application in blockchain design: replace "proof of work" with "proof of space"+proof of time
- More environment-friendly cryptocurrencies (e.g., Chia)

Questions?