Time-Lock Puzzles

Chethan Kamath, Pietrzak Group
Franke and Co

- **Protagonists**

  - **Franke**
  - **Miele**
  - **Jules**

Antagonists: Us
Franke and Co

- **Protagonists**
  - Franke
  - Miele
  - Jules

- **Antagonists: Us**
Motivation*

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

Cogito, ergo sum

2017

*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

*C* I shamelessly ripped this example off Tal Moran’s Crypto’11 talk.
**Motivation**

*Cogito, ergo sum*

Sic semper tyrannis!

*2017* ➔ *HELP!* ➔ *2042*

---

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

Cogito, ergo sum

Sic semper tyrannis!

2017

2042

2067

* 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

Cogito, ergo sum

Sic semper tyrannis!

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

Problem: Franke has to completely trust Miele Dishwashers break down

2017

HELP!

2042

HELP!
Attempt 1: Use a Trusted Third Party

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

\( n = 60: \approx 2^{32} \) years;
\( n = 128: \approx 2^{64} \) years
Franke and Jules share a key
Franke and Jules share a key
Franke and Jules share a key

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

Encrypt(message, key) = code
Encryption

- Franke and Jules share a key
- $\text{Encrypt}(\text{message}, \text{key}) = \text{code}$
Franke and Jules share a key
Encrypt(message, key) = code
Decrypt(code, key) = message
Encryption

- Franke and Jules share a key
- $\text{Encrypt}(\text{message}, \text{key}) = \text{code}$
- $\text{Decrypt}(\text{code}, \text{key}) = \text{message}$

- Key size: If key is $n$ bits then it takes $\approx 2^n$ operations on one computer to break the 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
- \( n = 60: \approx 25 \) years; \( n = 128: \approx 2^{32} \) years
Encryption...

Cogito, ergo sum

Sic semper tyrannis!

2017 2042 2067
Encryption...

Cogito, ergo sum

Sic semper tyrannis!

Start breaking 60 and 128 bit keys

2017 - 2042 - 2067
Encryption...

- 2017
- 2042
- 2067

Start breaking 60 and 128 bit keys

60-bit key broken
Encryption...

Start breaking 60 and 128 bit keys

60-bit key broken

128-bit key broken

2017

2042

2067

Apocalypse

$2^{32}$
Attempt 2: Use 60-bit Encryption

Humanity cannot decrypt in < 25 years

✓ 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

Humanity cannot decrypt in < 25 years

Jules can decrypt in 25 years
Attempt 2: Use 60-bit Encryption

Humanity cannot decrypt in < 25 years

✓ Jules can decrypt in 25 years

2017

2042
Attempt 2: Use 60-bit Encryption

Humanity cannot decrypt in < 25 years

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

Humanity cannot decrypt in < 25 years. Jules can decrypt in 25 years.

2017

2042
Attempt 2: Use 60-bit Encryption

Humanity cannot decrypt in < 25 years

✓ Jules can decrypt in 25 years
Attempt 2: Use 60-bit Encryption

Humanity cannot decrypt in < 25 years

Jules can decrypt in 25 years

2017

2042
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
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]

▶ \( \text{Time-Lock}(\text{message}, t) = \text{puzzle} \)
“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
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

Unlock

2017

2042

2067
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

  \[ 2 \rightarrow 2^2 \rightarrow 2^{2^2} \rightarrow \cdots \rightarrow 2^{2^{t-1}} \rightarrow 2^{2^t} \]
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:
    \[ 21 = 13 \times 1 + 8 \]
    \[ = 8 \% 13 \]
  - Addition modulo 13:
    \[ 7 + 8 = 15 \]
    \[ = 13 \times 1 + 2 \]
    \[ = 2 \% 13 \]
  - Multiplication modulo 13:
    \[ 6 \times 8 = 48 \]
    \[ = 13 \times 3 + 9 \]
    \[ = 9 \% 13 \]
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:

\[
21 = 13 \times 1 + 8 = 8 \bmod 13
\]
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:
    \[ 21 = 13 \times 1 + 8 \]
    \[ = 8 \mod 13 \]
  - Addition modulo 13:
    \[ 7 + 8 = 15 \]
    \[ = 13 \times 1 + 2 \]
    \[ = 2 \mod 13 \]
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:
    \[ 21 = 13 \times 1 + 8 \]
    \[ = 8 \% 13 \]
  - Addition modulo 13:
    \[ 7 + 8 = 15 \]
    \[ = 13 \times 1 + 2 \]
    \[ = 2 \% 13 \]
  - Multiplication modulo 13:
    \[ 6 \times 8 = 48 \]
    \[ = 13 \times 3 + 9 \]
    \[ = 9 \% 13 \]
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($message$, $t$) := ($message + 2^t \mod p$, $t$, $p$)
Attempt 1: Exponentiation modulo prime $p$

- Setting: Counting modulo large prime $p$ (i.e., group $\mathbb{Z}_p^*$)
- Time-Lock($message$, $t$) := ($message + 2^{2^t} \mod p$, $t$, $p$)
  - Naïve: $2 \mod p \rightarrow 2^2 \mod p \rightarrow 2^4 \mod p \rightarrow \ldots 2^{2^t} \mod p$

- Shortcut (using log($t$) squarings):
  1. $exp = 2^{t \mod (p-1)}$ (where $p-1$ is the group order)
  2. $2^{exp} \mod p$

- Unlock($puzzle$, $t$, $p$):
  1. $2^{2^t} \mod p$ using $t$ squarings
  2. $puzzle - 2^{2^t} \mod p$

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

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

- Time-Lock(\( message, t \)) := (\( message + 2^{2^t} \mod p, t, p \))
  - Naïve: \( 2 \mod p \rightarrow 2^{2^2} \mod p \rightarrow 2^{2^{2^2}} \mod p \rightarrow \ldots 2^{2^t} \mod p \)
  - Shortcut (using \( \log(t) \) squarings):
    1. \( \exp = 2^{t \mod (p - 1)} \) (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$(message, t) := (message + 2^{2^t} \mod p, t, p)$
  - Naive: $2 \mod p \rightarrow 2^2 \mod p \rightarrow 2^{2^2} \mod p \rightarrow \ldots 2^{2^t} \mod p$
  - Shortcut (using log$(t)$ squarings):
    1. $exp = 2^t \mod (p - 1)$ (where $p - 1$ is the group order)
    2. $2^{exp} \mod p$
Attempt 1: Exponentiation modulo prime $p$

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

- **Time-Lock**($message, t$) := ($message + 2^t \% p, t, p$)
  - Naïve: $2 \% p \rightarrow 2^2 \% p \rightarrow 2^2 \% p \rightarrow \ldots 2^t \% p$
  - 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: $2\% p \rightarrow 2^2\% p \rightarrow 2^2\% p \rightarrow \ldots 2^{2^t} \% p$
  
  - 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** $(message, t) := (message + 2^t \mod p, t, p)$
  - Naïve: $2 \mod p \rightarrow 2^2 \mod p \rightarrow 2^2 \mod p \rightarrow \ldots 2^t \mod p$
  - Shortcut (using $\log(t)$ squarings):
    1. $exp = 2^t \mod (p - 1)$ (where $p - 1$ is the group order)
    2. $2^{exp} \mod p$

- **Unlock** $(puzzle, t, p)$:
  1. $2^t \mod p$ using $t$ squarings
  2. $puzzle - 2^t \mod p$
Attempt 1: Exponentiation modulo prime $p$

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

- Time-Lock($message$, $t$) := ($message + 2^{2t} \pmod{p}$, $t$, $p$)
  - Naïve: $2 \pmod{p} \rightarrow 2^2 \pmod{p} \rightarrow 2^2 \pmod{p} \rightarrow \ldots 2^{2t} \pmod{p}$
  - Shortcut (using log($t$) squarings):
    1. $exp = 2^t \pmod{(p - 1)}$ (where $p - 1$ is the group order)
    2. $2^{exp} \pmod{p}$

- Unlock($puzzle$, $t$, $p$):
  1. $2^{2^t} \pmod{p}$ using $t$ squarings
  2. $puzzle - 2^{2^t} \pmod{p}$

- 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($message$, $t$) := ($message + 2^{2t} \pmod{p}$, $t$, $p$)
  - Naïve: $2 \pmod{p} \rightarrow 2^2 \pmod{p} \rightarrow 2^2 \pmod{p} \rightarrow \ldots 2^{2t} \pmod{p}$
  - Shortcut (using $\log(t)$ squarings):
    1. $exp = 2^{t}(p - 1)$ (where $p - 1$ is the group order)
    2. $2^{exp} \pmod{p}$
- Unlock($puzzle$, $t$, $p$):
  1. $2^{2^t} \pmod{p}$ using $t$ squarings
  2. $puzzle - 2^{2^t} \pmod{p}$

- 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?