Time-Lock Puzzles
In the Random Oracle Model

Mohammad Mahmoody, Tal Moran, Salil Vadhan
Time-Lock Puzzles

- Sending an encrypted message to the future
  - shouldn’t be revealed before some future date
  - no safe storage for secrets

- Encode key as a “time-lock” puzzle
  - Bounds for computation time to solve puzzle
    - e.g., can be solved in 25 years on reasonable computer
    - Requires at least 20 on today’s fastest computer
  - Puzzle generation is fast

Also useful for:
fair contract signing, sealed-bid auctions, coin flipping and more [RSW96,BN00,...]
Naïve Puzzle

- Invert a one-way function
  - Give some of the input to reduce search space
  - (Assume brute-force is the only attack)
    \[ y = f(x_1, x_2, \ldots, x_{100}), x_1, x_2, \ldots x_{50} \]

- Attackers might have many more computers!
  - e.g., Botnets, “cloud” servers.
    - Shouldn’t gain a large advantage over legitimate solver (with one computer)

- Want a puzzle that is inherently sequential
Known Solution \([RSW96]\)

- Exponentiation (modulo \(N\))
  \[ f(x) = 2^{2^x} \mod N \]
  - Fastest known method is repeated squaring
    - takes \(\Omega(x)\) time
  - Can solve puzzle quickly if \(\varphi(N) = (p-1)(q-1)\) is known
    - compute \(x' = 2^x \mod \varphi(N)\)
    - compute \(2^{x'} \mod N\)

- Requires RSA assumption
  - what about quantum botnets?
  - Can we use other assumptions?
The Random Oracle Model

- Answer to each query is uniformly random (independently of other queries)
- The same query always gets the same answer
- **Complexity: count # of queries**
- Random Oracle is one-way even for computationally unbounded players
  - Impossibility results in RO rule out black-box constructions in standard model
- Heuristic for converting RO protocols to standard model
  - Replace RO with cryptographic hash (e.g. SHA256)
  - Not provably secure, but is used in practice
Our Results: Overview

• Main Result:
  – Time-lock puzzles that require \( n \) queries to generate can be solved in \( n \) parallel steps.
  – Rules out black-box constructions from one-way/hash functions

• Positive result:
  – Simple Time-lock puzzle satisfying
    • \( n \) parallel queries to construct
    • \( n \) sequential queries required to solve

(totol # queries polynomial in honest solver)

Generator with \( n \) parallel CPUs - \( n \) times faster than solver
Main Result

Based on ideas from attacks on key-exchange protocols in the random oracle model [IR89,BM09]
Main Result

- High-level Sketch:
  - Construct adversary that finds intersection queries
Main Result

• High-level Sketch:
  – Construct adversary that finds intersection queries
  – Run honest solver with simulated oracle
    • Answer known queries correctly, others randomly
  – Success prob. identical to honest solver
  – Main hurdle: find intersections with low adaptivity

From generator’s point of view, “real” answers are identical to “fake” on unqueried indices
Finding Intersection Queries

(efficient adversary with non-optimal adaptivity)

- For all $\varepsilon$, adversary uses $\frac{n}{\varepsilon}$ rounds of queries
  - Queries in each round can be done in parallel
- In each round:
  - Simulate honest solver
  - Answer known queries correctly, others randomly
  - Ask all queries to real oracle in parallel after every round
- Output results of randomly chosen round
Finding Intersection Queries: Analysis

• Success probability: $1-\varepsilon$
  – If simulation in output round did not hit any new intersection queries:
    simulated output is identically distributed to honest output (success probability is 1)
  – Generator asks at most $n$ queries
    • Adv. asks a new intersection query in at most $n$ rounds
  – Random round hits all intersection queries with prob. $1-\varepsilon$

• Query complexity: $nm/\varepsilon$

• Computational complexity:
  – polynomial in honest solver complexity
Positive Construction

- Time-lock puzzle encodes “pointer chain”
  - Generator queries in parallel
  - Solver must serially follow pointers

If adversary does not query oracle, it cannot do better than guessing next pointer
Discussion and Open Questions

- Optimally Adaptive (but inefficient) adversary
  - Uses $n$ rather than $n/\varepsilon$ adaptive rounds
  - Based on new learning algorithm for intersection queries.
- Corollary:
  - “Merkle puzzles” can be solved in linear parallel time
- Our negative result does not rule out “proofs of work”
  - In a proof-of-work, puzzle generator can verify solution quickly but not solve.
  - Positive solutions exist (work in progress)
- Still open:
  - Other time-lock puzzles in standard model?
  - Time-lock puzzles for quantum computers?
    - Related to [BHKKLS11] (coming soon to a lecture hall near you!)