BITCOIN MINING IN A SAT FRAMEWORK

Jonathan Heusser

@jonathanheusser
DISCLAIMER

JUST TO BE CLEAR..

This is research! Not saying ASICs suck
I am not a cryptographer, nor SAT solver guy
WTF
REALISED PHD RESEARCH CAN MINE BITCOINS
Phd in static analysis + information theory
Quantifying information leakage in programs

Same techniques can be used for mining without brute force!
## BLOCK HEADER

<table>
<thead>
<tr>
<th>Field Size</th>
<th>Description</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>version</td>
<td>uint32_t</td>
<td>Block version information, based upon the software version creating this block</td>
</tr>
<tr>
<td>32</td>
<td>prev_block</td>
<td>char[32]</td>
<td>The hash value of the previous block this particular block references</td>
</tr>
<tr>
<td>32</td>
<td>merkle_root</td>
<td>char[32]</td>
<td>The reference to a Merkle tree collection which is a hash of all transactions related to this block</td>
</tr>
<tr>
<td>4</td>
<td>timestamp</td>
<td>uint32_t</td>
<td>A timestamp recording when this block was created (Will overflow in 2106[2])</td>
</tr>
<tr>
<td>4</td>
<td>bits</td>
<td>uint32_t</td>
<td>The calculated difficulty target being used for this block</td>
</tr>
<tr>
<td>4</td>
<td>nonce</td>
<td>uint32_t</td>
<td>The nonce used to generate this block... to allow variations of the header and compute different hashes</td>
</tr>
</tbody>
</table>
MINING CORE
GETBLOCKTEMPLATE

template = getblocktemplate()
while extranonce < MAX:
    block_header = create(template, extranonce)

    while nonce < MAX:
        if f(block_header) < target:
            return 'Found valid block'
        nonce++
    extranonce++

nonce and extranonce pointers into block_header
MINING CORE
MINERS FOCUS ON BRUTEFORCE

```python
template = getblocktemplate()
while extranonce < MAX:
    block_header = create(template, extranonce)

    while nonce < MAX:
        if sha2(sha2(block_header)) < target: // f(x
            return 'Found valid block'
        nonce++
extranonce++
```

f is considered a blackbox, not part of algorithm
brute force, because no method or logic involved
no connection between f and nonce
PROPERTY

AVAKALANCH EFFECT

Good hash: 1 bit flipped in input, a lot of bits touched in output

Observing the output of a hash function tells you nothing about input

Output uniformly distributed no matter what input distribution

If that was not the case: search possible by playing with nonce
Make f run as fast as possible!

Pools, GPU, FPGA, ASIC
IN THIS TALK

Connect f and nonce!

Using tools from program verification: model checker and SAT solver
IN THIS TALK

Connect f and nonce! No brute force

Using tools from program verification: model checker and SAT solver

---

Build declarative specification for mining

Model specification using model checking

Solve for nonce using SAT solver
DECLARATIVE SPECIFICATION (VS IMPERATIVE ALGO)

```plaintext
nonce = * // don't care the actual value! Any value!
hash = sha2(sha2(block_header))
assume(hash < target) // constraint
```

---

Specification for set of valid mining solutions
Here, f and nonce connected through assumption and global constraint
---

How to encode and solve?
MODEL CHECKING

FORMAL VERIFICATION USING MODEL CHECKING

Extremely successful in practice but not well known (Turing Award)

CPU designs, avionics, medical apps only safe due to verification

Given system, check exhaustively properties of that system

Provide counter example to violation of property

Example property: absence of deadlocks, floating point errors, etc
BOUNDED MODEL CHECKING

VERIFICATION OF PROGRAMS IS HARD

State explosion: trivial program has infeasible number of states

Abstraction or restriction of power necessary

Bounded model checker is only a bug hunting tool. Bounding loops
BOUNDDED MODEL CHECKING

VERIFICATION OF PROGRAMS IS HARD

State explosion: trivial program has infeasible number of states

Abstraction or restriction of power necessary

Bounded model checker is only a bug hunting tool. Bounding loops

CBMC bounded model checker translates C to logic and hunts for bugs

bug means specification violation
BOUNDDED MODEL CHECKING

C TO PROPOSITIONAL LOGIC

Bitvector variables, unrolled loops, SSA form, ...

Semantics mostly preserved

Program is one global constraint

\[
\begin{align*}
\text{if}(x_1 = 1) & : x_2 = 2; \\
\text{else} & : x_2 = x_1 + 1; \\
\text{assert}(x_2 \leq 3); \\
\text{end}
\end{align*}
\]
PROPERTY CHECKING

\[ C \land \neg P \]
Passed to decision procedure. Only satisfiable IFF property
P violated
Counterexample: execution path to violation of P
SAT SOLVING

DECISION PROCEDURE:
SATISFIABILITY SOLVER

Decide whether logic formula has a solution (is satisfiable)

Very active and competitive research area

Solvers based on Davis–Putnam–Logemann–Loveland
(DPLL) algorithm

Extremely efficient: 100k's vars, millions of clauses
SAT SOLVING

CONJUNCTIVE NORMAL FORM (CNF)

Formula in CNF: 'ands of ors'

\[(\neg a \lor b) \land (\neg a \lor c) \land (\neg b \lor \neg c \lor d)\]

For each clause, at least one literal true

All clauses true in order to be SAT
SAT SOLVING

DPLL ALGORITHM

Depth-first search by picking literals
Propagate decision
Backtrack on conflict

Lots of variations and heuristics
SAT SOLVING

SAT AND CRYPTOGRAPHY

Many papers on using SAT solvers for attacking ciphers
Represent cipher as equations, solve using SAT

Special solvers with XOR, Gauss elimination, variable
activity support, ..
Cryptominisat (Mate Soos)
SAT-BASED MINING

ENCODE SPECIFICATION USING CBMC

Translate specification into C code
Annotate with CBMC specific assumptions and assertions
nonce = nondet_int()
hash = sha2(sha2(block_header))
assume(hash[0] == 0 && hash[1] == 0 && ..)  // assu
assert(hash > target)                     // prop

---

Nonce is a non-deterministic value

Known structure of valid hash: leading zeros are assumed

Assertion that valid nonce does not exist
void satcoin(unsigned int *block) {
    unsigned int *nonce = block+N;
    *nonce = nondet_int();

    // 'sha' is a standard SHA-256 implementation
    hash = sha(sha(block));

    // assume leading zeros
    assume(hash[0] == 0x00 && ...);

    // encode a state where byte M of hash is bigg
Demo Time
COMPARISON

SAT VS BRUTEFORCE

Clearly, brute force much faster. Only direction is making f
faster though

Encode richer specification: leading zeros, tricks in SHA2,
set individual bits in nonce, ...

Specialised SHA2 encoding: Vegard Nossum, sha256-sat-
bitcoin

Take advantage of SAT solvers: learnt clauses, variable
activity, cryptominisat, portfolio solvers
COMPARISON

INCREASING DIFFICULTY

Increasing difficulty results in more leading 0 in hash
Conceptually restricts search space
Does this lead to more efficient SAT solving?
REFERENCES

SOME RELEVANT PAPERS

SAT Solving - An alternative to brute force bitcoin mining
SAT-based preimage attacks on SHA-1
The Unreasonable Fundamental Incertitudes Behind Bitcoin Mining
Algebraic Fault Attack on the SHA-256 Compression Function
THANK YOU