# APPLYING SATISFIABILITY TO THE ANALYSIS OF CRYPTOGRAPHY

Aaron Tomb, Galois, Inc.

September 25, 2015

- Cryptographic algorithms are critical
  - Central to commerce, private communication, etc.
  - We want them to be correct

- Cryptography is hard
  - Rare expertise
  - Top experts have designed algorithms now known vulnerable
  - Design $\rightarrow$ implementation can introduce other problems

- Key primitives used in cryptography amenable to SAT
  - Can be given denotational semantics in propositional logic
  - Many interesting problems surprisingly tractable

|galois|

- Many primitives naturally made of bit vector operations
  - Block ciphers
  - Stream ciphers
  - Hash functions
  - Pseudo-random number generators (PRNGs)

- Public-key algorithms representable, but trickier
  - Lots of number theory, including multiplication
  - SMT can alleviate some of this (but tends to be slower on bit-vectory things)

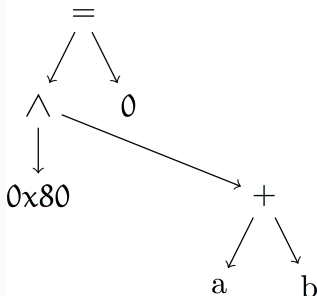- We'll focus on primitives with type $\{0, 1\}^n \rightarrow \{0, 1\}^m$

galois

- A function from the ZUC stream cipher

```
uint8_t c = a + b;
if (c & 0x80) {
  c = (c & 0x7F) + 1;
}
return c;
```

- A function from the ZUC stream cipher

```
uint8_t c = a + b;
if (c & 0x80) {
  c = (c & 0x7F) + 1;
}
return c;
```
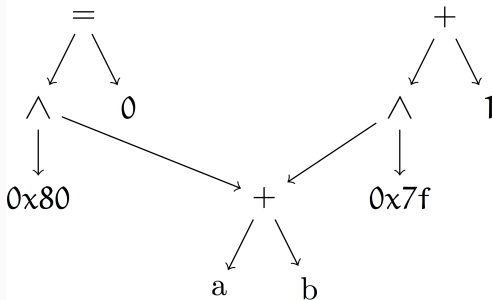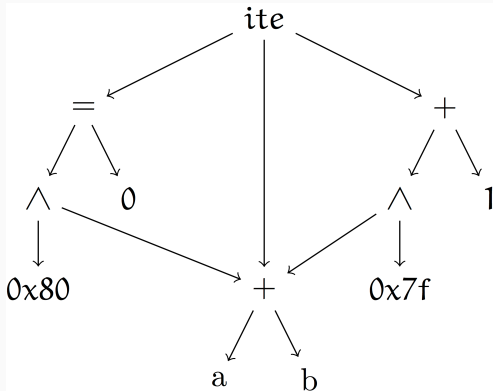
- A function from the ZUC stream cipher

```
uint8_t c = a + b;
if (c & 0x80) {
  c = (c & 0x7F) + 1;
}
return c;
```

- A function from the ZUC stream cipher

```
uint8_t c = a + b;
if (c & 0x80) {
  c = (c & 0x7F) + 1;
}
return c;
```
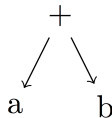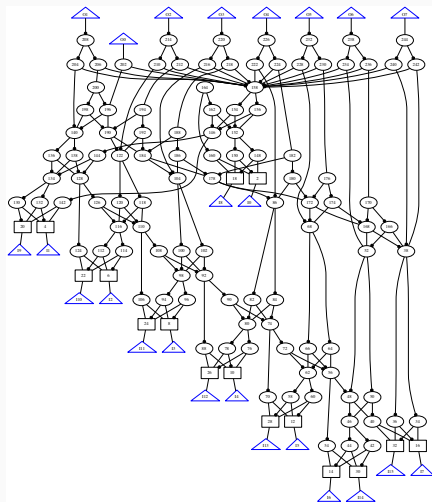
- Transalg (Russian Academy of Sciences)
- Cryptol and the Software Analysis Workbench (SAW) (Galois)
  - Cryptol for describing algorithms concisely
  - SAW for proving properties about them
- TODO: others

- Each with a concise formula summarizing the SAT problem

$$\forall x.\ P \text{ or } \exists y.\ Q$$

- Cryptol specifications available online
- Analyzed using SAW + ABC
- Cryptol or SAW file names mentioned in slides (e.g. [file.saw])

https://github.com/galoisinc/sat2015-crypto

- Randomness is critical to cryptography
    - Keys must be unpredictable, or they're vulnerable
- Typical structure:

    seed $\rightarrow$ pseudo-random function $\rightarrow$ pseudo-random value

- Seed often comes from "true" random source
- But it's critical not to lose entropy from this source!

- Bug in Android cryptographic PRNG discarded entropy
- Intended: 160 bits of entropy, from system PRNG to SHA-1

| 0 | 160 | 224 | 255 |
|---|---|---|---|
| seed | | counter | 0x80000000 |

- Implemented: 64 bits of entropy survive

| 0 | 64 | 96 | 160 | 255 |
|---|---|---|---|---|
| counter | 0x80000000 | seed (rest) | 0 . . . 00 | |

- Used to steal around $6k of BitCoin
- Fixed in Android 4.4 (KitKat)
- Similar weaknesses existed in Debian, FreeBSD at times

|galois|

$$\forall x, y.\ x \neq y \Rightarrow f(x) \neq f(y)$$

- Here, $x$ and $y$ come from system PRNG, $f$ is code between system PRNG and SHA-1
- Easily provable with SAT solver [android-prng.saw]
    - Symbolic execution: Java code $\rightarrow$ AIG
    - Annotation on system RNG variables, input to SHA-1
    - ABC can find collision (0.017s), prove fixed version (0.010s)
- Dörre and Klebanov used a different approach to prove fixed code
    - Information flow annotations on methods using a contract verification tool (KeY) and 95 manual proof steps

|galois|
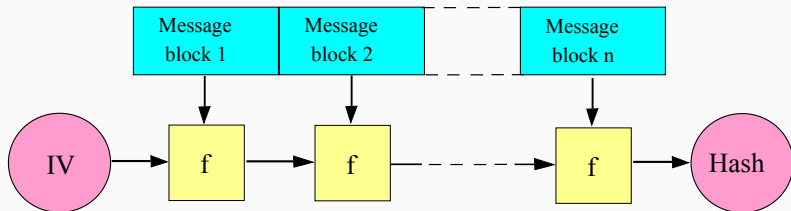
- Key property: hard to find two messages that have the same hash (a collision)
- Often built using Merkle–Damgård construction, iterating compression function
- Compression function f usually $n$ iterations of simpler function g

|galois|

$$\exists x, y. \; x \neq y \wedge f(x) = f(y)$$

- Discovering a collision
- Black box search in $O(2^{n/2})$

$$\exists x. \; f(x) = a \text{ (for some known value } a)$$

- Discovering message given hash value (inversion)
- Harder than finding a collision ($O(2^n)$)
- We want to know that it's hard to solve these problems

|galois|

- Mironov and Zhang analyzed collisions in MD4, MD5, SHA-0

  - Direct translation of MD4 $\rightarrow 2^{22}$ solutions in $< 1$h
  - Collision on full MD5 in around 100h

    - Reduced rounds much easier (25 in 38s with ABC) [MD5.saw]
    - Used differential path derived manually (more on this later)

  - Estimated around 3 million (2006-era) CPU hours for SHA-0

- No known collisions on SHA-1 from this, but it may be a matter of time

  - Algorithm (not SAT-based) to find collisions in $2^{63}$ operations

- Many candidate algorithms for the SHA-3 standard
- No preimages discoverable by SAT on full algorithms
- Homsirikamol et al. found preimages for fewer rounds
  - Direct translation, with no manual cryptanalysis

| Algorithm | Rounds | Security margin | Code |
| --- | --- | --- | --- |
| SHA-1 | 21 | 74% (21/80) | SHA-0-1.saw |
| SHA-256 | 16 | 75% (16/64) | SHA265.saw (TODO) |
| Keccak-256 | 2 | 92% (2/24) | N/A |
| BLAKE-256 | 1 | 93% (1/14) | blake256.saw (TODO) |
| Groestl-256 | 0.5 | 95% (0.5/10) | N/A |
| JH-256 | 2 | 96% (2/42) | N/A |
| Skein-512-256 | 1 | 99% (1/72) | N/A |

| galois |

- Given a key, a block cipher is a pseudo-random permutation
  - Therefore, invertable (for a fixed key)
- Often built as a substitution-permutation network
  - Will return to S-boxes
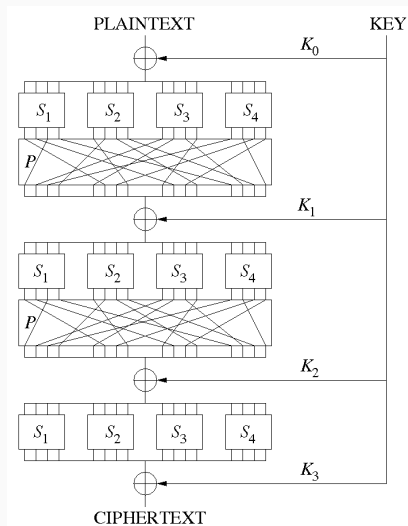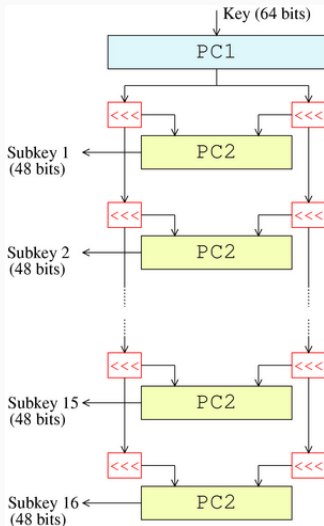
- Symmetric encryption often involves a single shared key
    - Block ciphers typically need one key per round
    - Stream ciphers need one "key" per message block

- So we need to expand the initial key in an unpredictable way
- Should preserve size of key space

*galois*

$$\forall x, y.\ x \neq y \Rightarrow f(x) \neq f(y)$$

- Provable injectivity of SIMON, Salsa20 key expansion
  - Several seconds with ABC [simon.saw] [Salsa20.saw]
- Provable injectivity of AES key expansion
  - A little under a minute with ABC [AES.saw]
- ZUC, a stream cipher for GSM, had a vulnerability
  - Key expansion not injective in ZUC 1.4 (0.5s) [zuc.saw]
  - Provably fixed in ZUC 1.5 (0.6s) [zuc.saw]
  - Originally shown with custom search procedure taking 3m
- Not injective for DES!
  - Can show quickly with ABC (0.08s) [DES.saw]
  - Unaware of a published attack that uses this fact (TODO)

| galois |

$$\exists m. \; E(m, k) = c \text{ for known } k \text{ and } c$$

- Decrypting using encryption code
- This actually works!
  - Relatively efficient for DES (0.2s), 3DES (0.8s), SIMON (TODO: time), Speck (TODO: time) [DES.saw] [3DES.saw] [simon.saw]
  - Takes ~1.5min for one block of AES [aes.saw]

- Can also run the encryption directly:
  - $\exists c. \; E(m, k) = c$ for known $m$ and $k$
  - Also around ~1.5min for AES [aes.saw]
  - Not really useful, but illustrates the flexibility of SAT

|galois|

$$\forall m, k. \ D(E(m, k), k) = m$$

- For a block cipher with encryption function $E$, decryption $D$
- Easy to show for many ciphers

  - DES (4s) [DES.saw]
  - 3DES (8s) [3DES.saw]
  - SIMON (128-256) (6.2m) [simon.saw]
  - Speck (TODO: time) [speck.saw]

- Hard to show for AES (at least with encodings and solvers we've tried) [AES.saw]

$$\forall x.\; f(x) = g(x)$$

- Two functions give equivalent output for all inputs
- AIGs give distinct benefits over direct CNF creation
  - Intuitive construction
  - Sharing subterms reduces overall expression size
  - SAT sweeping helps identify candidate equivalences
    - Especially effective for cryptography
  - Use best available SAT solver for final phase
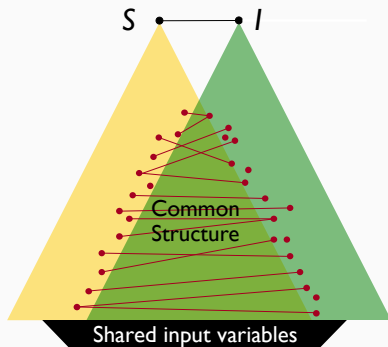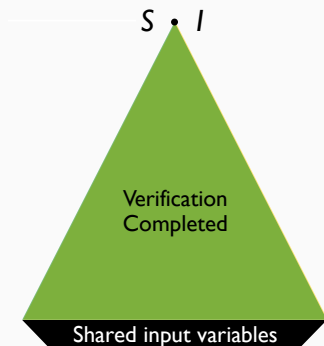- Works on many cryptographic primitives, including AES (~6m) [AES-eq.saw]
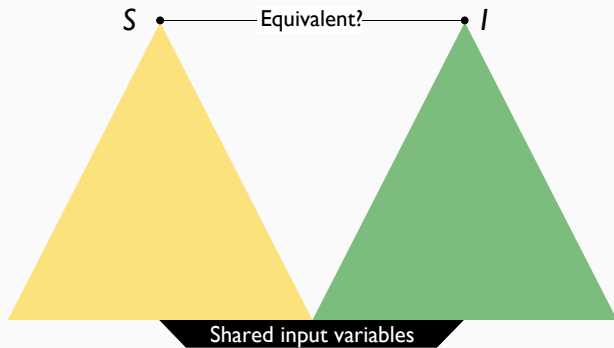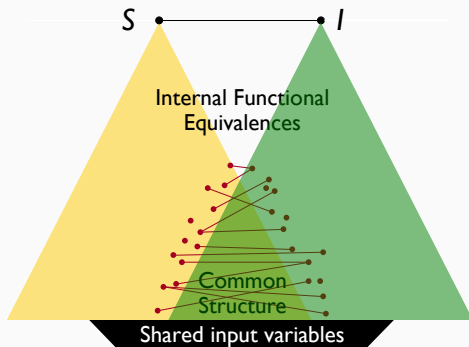
|galois|
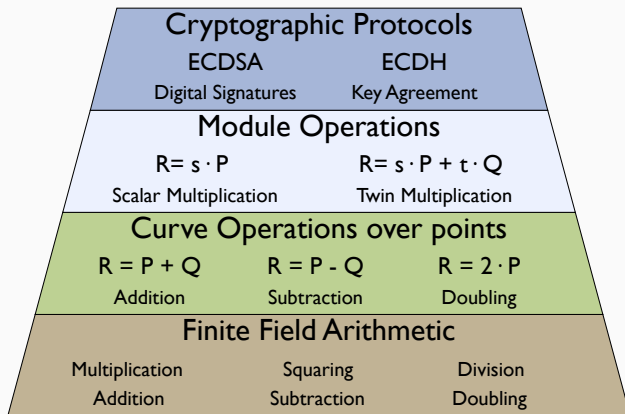
- Key tool: uninterpreted functions
- Symbolic execution turns imperative code into functional code
  - So procedure calls can be uninterpreted functions
  - If we know all inputs and outputs
- Used for checking equivalence between Cryptol and Java ECDSA
  - Takes around 5 minutes to run
  - Takes ~1500 lines of script (mostly I/O mapping, a few rewrite rules)
  - ABC for leaves, rewriting + Z3 for higher layers

|galois|

- Attack on symmetric block ciphers
- Known plaintext attack: assumes attacker has some set of $(m, c)$ pairs
- Basic idea: can we approximate the encryption function by a linear function?
    - Where *linear* here means made up entirely of XOR operations
- Any time the encryption function agrees with a linear function too often, this can ease cryptanalysis
- Can use #SAT to count how often it behaves linearly

- Like linear cryptanalysis, known plaintext attack on symmetric block ciphers
- Analysis of how differences in input affect differences in output
- Disproportional effects can be exploitable
- A differential characteristic is a set of differences as they traverse a path through the algorithm
- Can use #SAT to calcuuate the distributions of differential characteristics

galois

- SIMON is a lightweight block cipher published recently
- Kölbl *et al.* used SAT for linear and differential cryptanlysis of SIMON [2] [simon-diff.saw]
- Using #SAT to calculate differential characteristic distributions
- Not direct analysis of code, but of manually-derived simplification
- Serves as an additonal tool for cryptanalysis, not push-putton
- Suggests slightly different parameters possibly preferable to published numbers
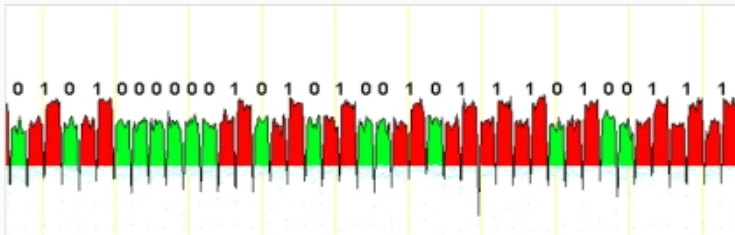
- Traditional side channel analysis
  - Observe specific part of last round of block cipher
  - Combine with pre-calculated formulas to recover key
- Caveat: only works if you can observe the right signals

|galois|

- Example: hardware implementation
  - Encode entire algorithm (as implemented) as a boolean circuit
  - Observe any internal values possible
  - Constrain internal variables accordingly
  - Try to solve for key (or other aspects of state)

- More information than just inputs or outputs
- Can use any internal variable in the algorithm
- Hamming weights plus a handful of $(m, c)$ pairs enough to recover AES key [3]

|galois|

- S-boxes intended to unpredictably substitute bits
  - Ideally indistinguishable from a random function
  - But in practice representable as a short program using only linear operations

- Generating optimal S-boxes:
  - Or any function on a small domain
  - $\exists p.\ [\![p]\!](x_0) = y_0 \wedge \cdots \wedge [\![p]\!](x_n) = y_n$

- Fuhs *et al.* found a 23-instruction program for the AES S-box [1]
  - Less than a minute with MiniSat
  - Proving unsatisfiability of 22 instructions took 106 hours (CryptoMiniSat)

|galois|

- General synthesis:
  - $\exists p. \forall x. [\![p]\!](x) = f(x)$

- Generating efficient implementations:
  - $\min p. \forall x. [\![p]\!](x) = f(x)$

- Generally, a hard problem
  - But QBF solvers are getting powerful
  - Many papers in SMT community about this problem recently

- Cryptographic protocol analysis
- Diffusion analysis
- Differential fault analysis
- TODO: more!

- Some problems are difficult with current solvers, but potentially tractable
  - AES equivalence in CNF (tractable using SAT sweeping!)
  - AES consistency
  - Good benchmarks for solver improvement!
- Some problems that are intrinsically difficult (we hope)
  - Multiplication difficulty seems related to hardness of factoring
  - Finding a hash collisions had better be hard
  - Finding $k$ given $m$, $c$ should be hard, too
- Synthesis is currently hard
  - But provers are getting better at this (QBF and $\exists\forall$-SMT)
- Very similar problems can be different in difficulty
  - Reversing AES vs. finding $k$ from $m$, $c$

|galois|

- Cryptol is a DSL for cryptographic code
  - Allows expression of algorithms at a high-level of abstraction
  - Built-in connection to SMT solvers
  - BSD-licensed
    - http://cryptol.net

- The Software Analysis Workbench (SAW) allows analysis of implementations
  - Symbolic execution of Cryptol, C, Java
  - Bindings to SMT and SAT solvers, including ABC
  - Freely available (with source) for non-commercial use
    - http://saw.galois.com

- All the examples from this talk are available
  - https://github.com/galoisinc/sat2015-crypto

| galois |

- Cryptography and SAT go very nicely together
    - Easily representable as propositional formulas
    - AIGs are super handy! (not just for hardware)

- More use of SAT during algorithm development will make our crypto stronger
    - And it's happening: see SHA-3, Trivium, SIMON, etc.

- Nice source of hard benchmark problems for solvers

    https://github.com/galoisinc/sat2015-crypto

| galois |

[1] Fuhs, C. and Schneider-Kamp, P. 2010. Optimizing the aES s-box using sAT. *IWIL 2010. the 8th international workshop on the implementation of logics* (2010), 64–70.

[2] Kölbl, S. et al. 2015. Observations on the sIMON block cipher family. *Advances in cryptology – cRYPTO 2015*. R. Gennaro and M. Robshaw, eds. Springer Berlin Heidelberg. 161–185.

[3] Mohamed, M. et al. 2012. Improved algebraic side-channel attack on aES.