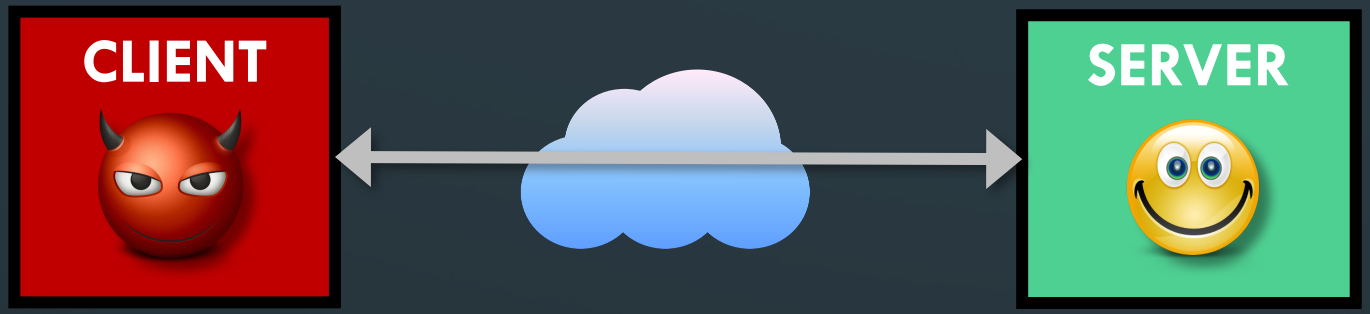
A SYSTEM TO VERIFY NETWORK BEHAVIOR OF KNOWN CRYPTOGRAPHIC CLIENTS

Andrew Chi, Robert A. Cochran, Marie Nesfield, Michael K. Reiter, Cynthia Sturton

University of North Carolina at Chapel Hill

INVALID COMMAND ATTACKS

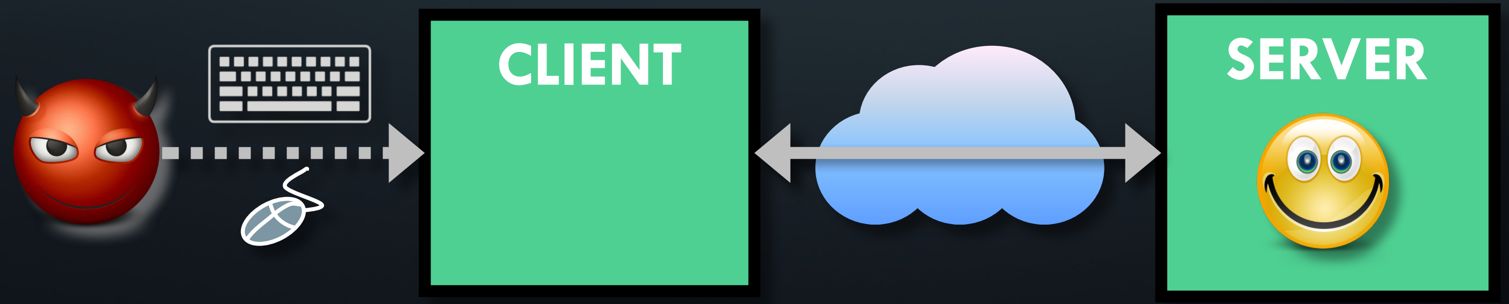


INVALID COMMAND

Client exhibits behavior, as seen by the server, that is inconsistent with the sanctioned client software.

Forms of Exploit:

1. **Maliciously crafted packet**
2. **Valid packets; illegal sequence**



Constrained attacker
(valid commands only)

Goal: constrain *all attackers* to this limited behavior.

TRANSPORT LAYER SECURITY (TLS) - RFC 5246

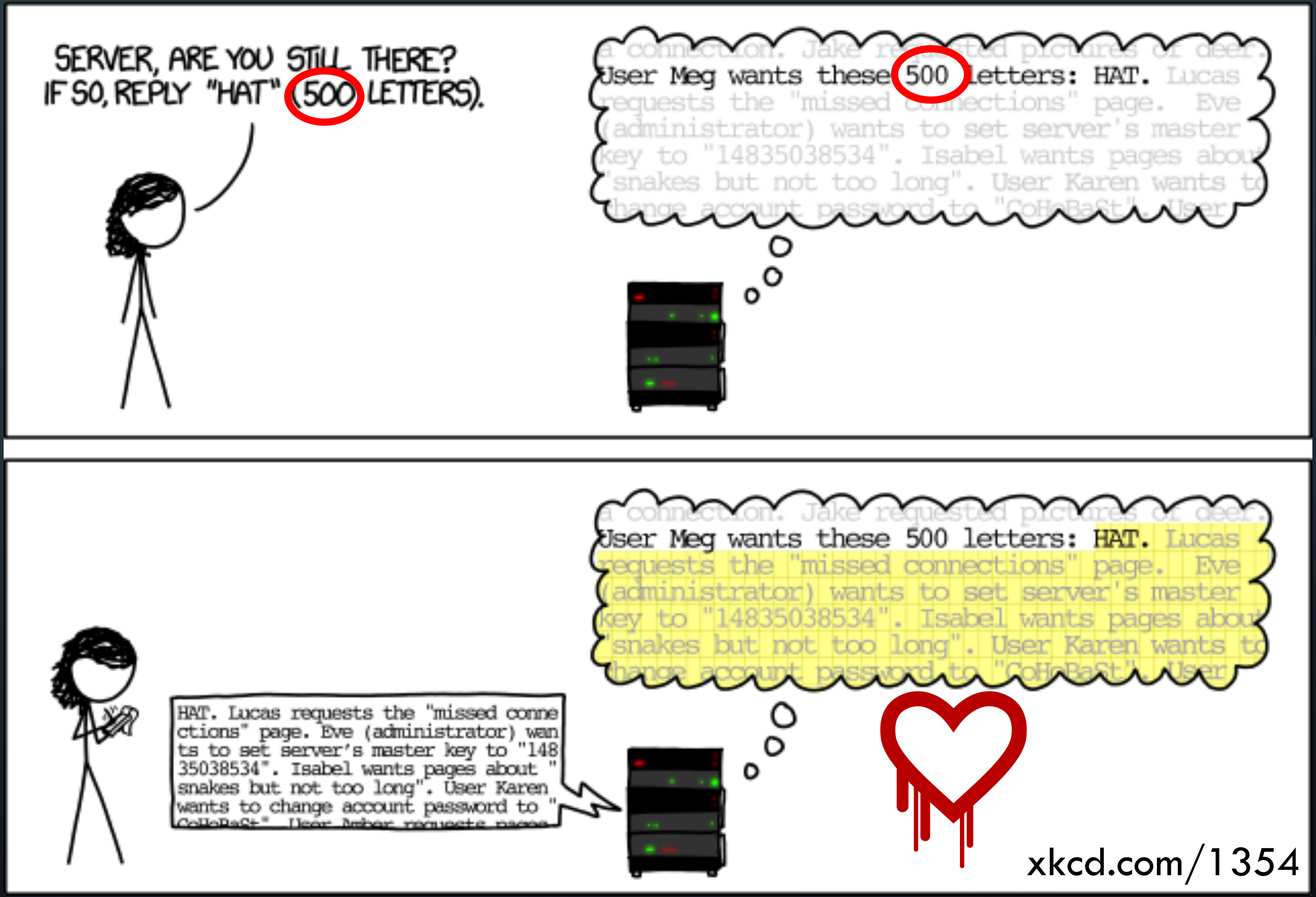
- Handshake Protocol
  - Select cipher, authentication, key exchange
- Record Layer
  - Provides confidentiality and integrity
  - Encapsulates other protocols
- Alerts and Heartbeats

From Jan 2014 to Aug 2016, most of the server-side vulnerabilities in OpenSSL involved invalid commands (23 of 37 required tampering with client behavior).

HEARTBLEED (CVE-2014-0160)

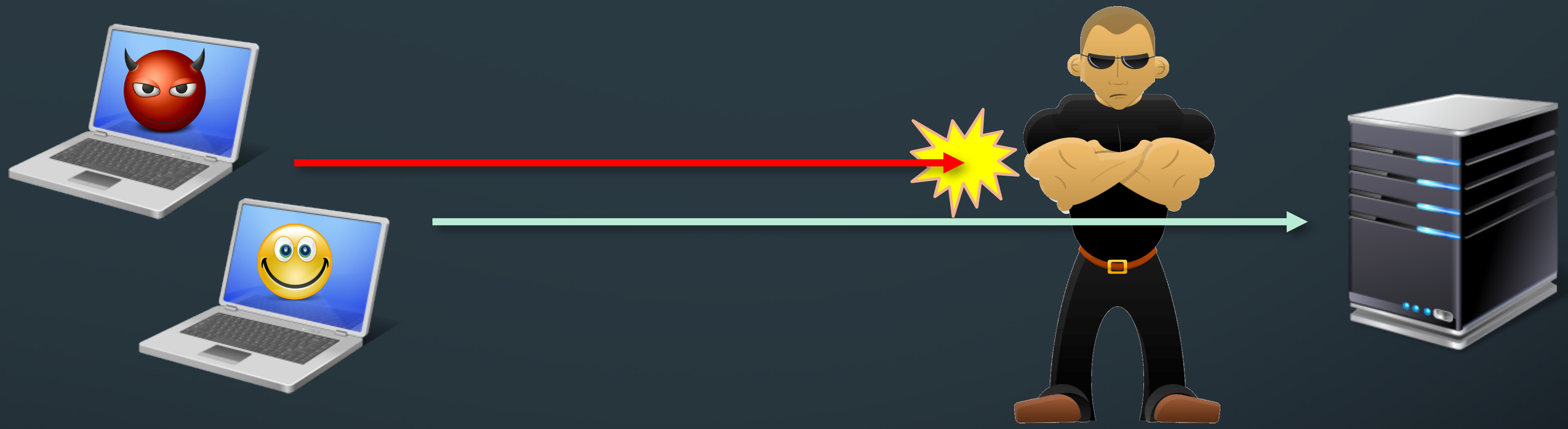
- Implementation bug in OpenSSL TLS Heartbeat handler
- Nearly all OpenSSL applications vulnerable for 2 years
- 17% (~500,000) of the Internet's web servers

HEARTBLEED



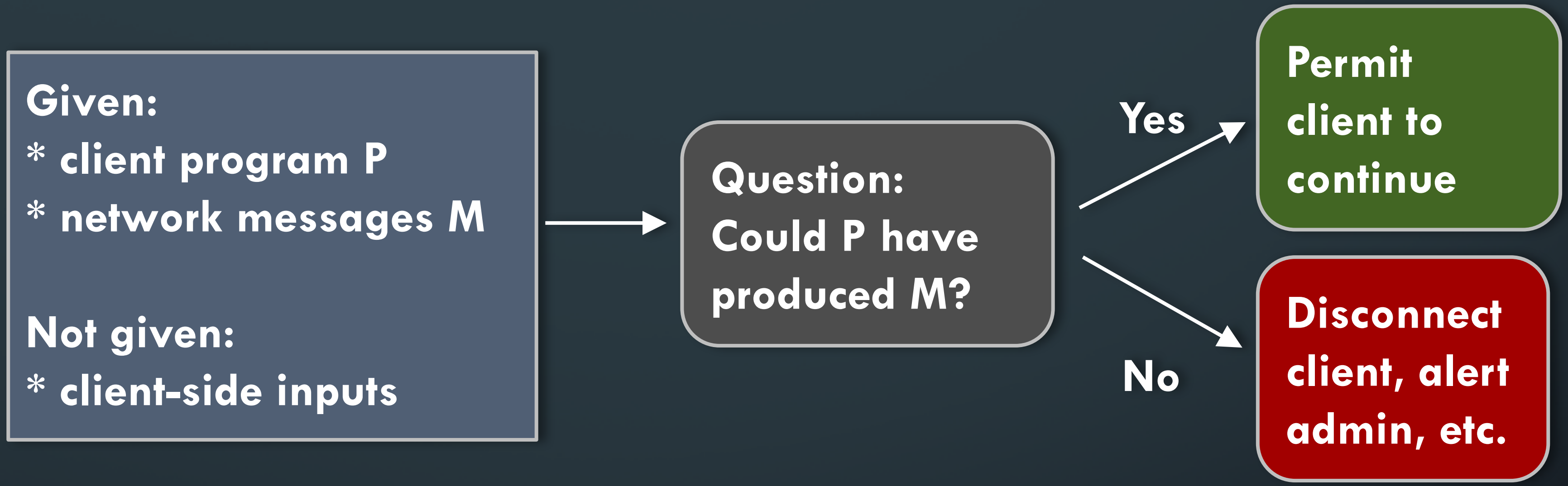
xkcd.com/1354

HOW CAN WE DEFEND THE SERVER?



- Behavioral verification: permit authorized client software's behavior only
  - Eliminates entire classes of attack *without knowing about them*
  - Usually requires client modification or sending of client inputs
- Goal: rapid detection of exploit attempts

BEHAVIORAL VERIFICATION OF A CLIENT

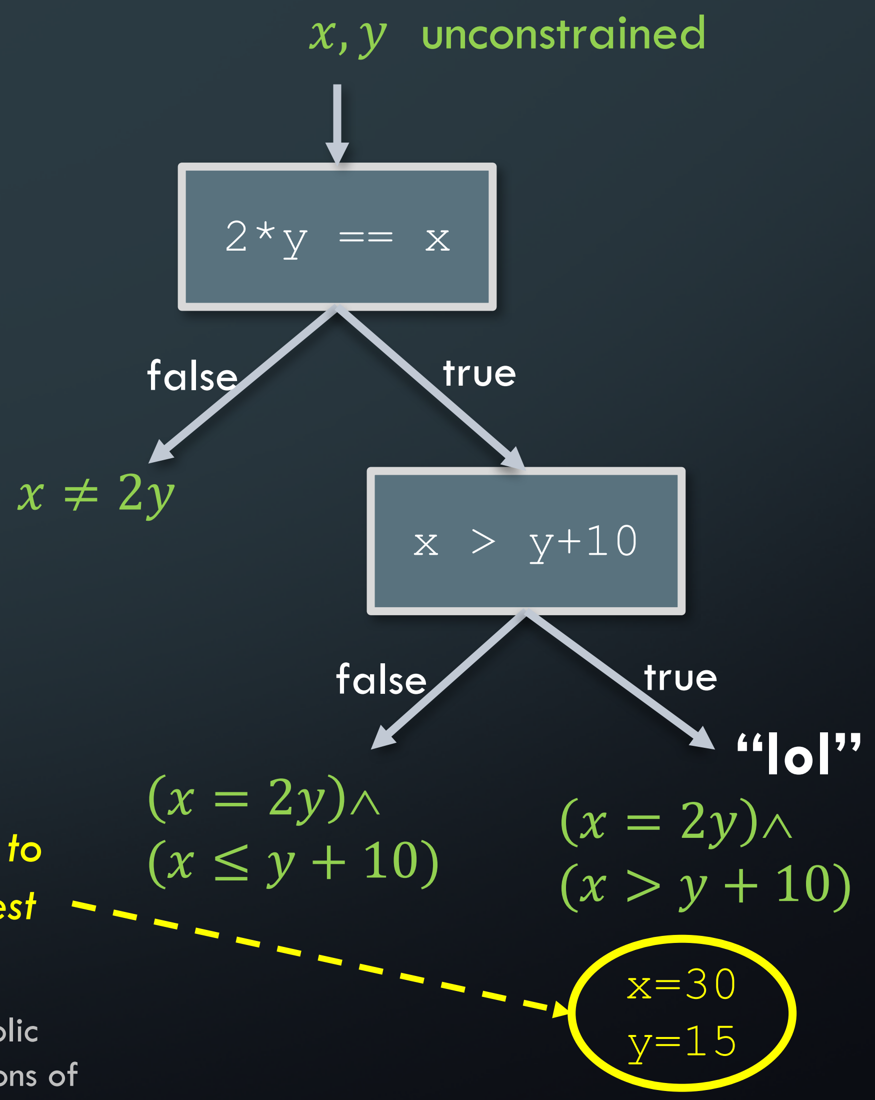


- General case: undecidable
- Specific instances: may be practical
- E.g., detect cheating in online games (Cochran & Reiter 2013)

SYMBOLIC EXECUTION

```
x = sym_input();
y = sym_input();
testme(x,y);

void testme(int x, int y)
{
    int z = 2*y;
    if (z == x) {
        if (x > y+10)
            printf("lol");
    }
}
```



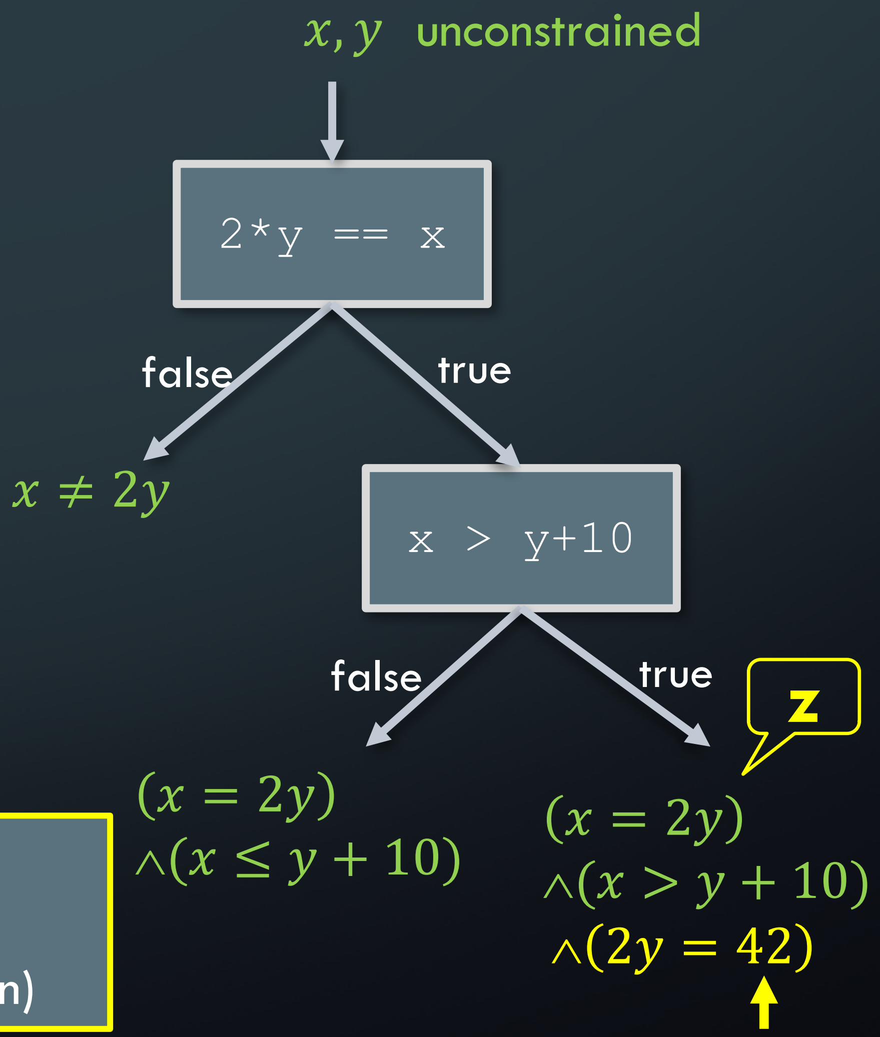
Apply SAT solver to obtain concrete test case.

Example adapted from: Cristian Cadar, and Koushik Sen. "Symbolic execution for software testing: three decades later." Communications of the ACM 56.2 (2013): 82-90.

USING SYMBOLIC EXECUTION TO DETECT INVALID COMMAND ATTACKS

```
x = sym_input();
y = sym_input();
testme(x,y);

void testme(int x, int y)
{
    int z = 2*y;
    if (z == x) {
        if (x > y+10)
            send(z);
    }
}
```

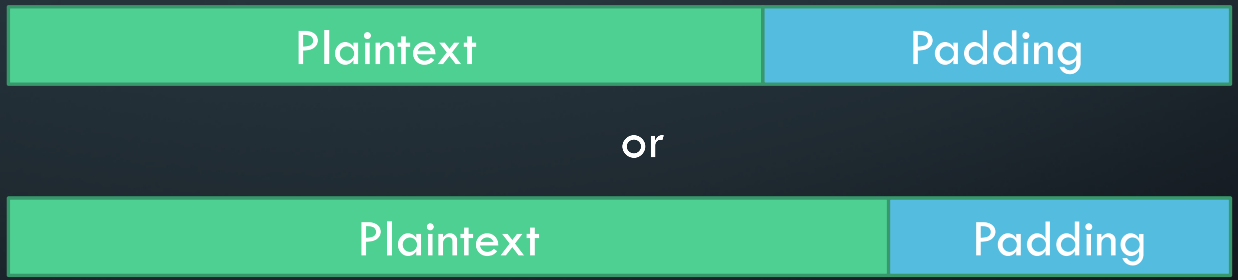


Can this program produce…

- $z = 42$**?** **Yes** $(x = 42, y = 21)$
- $z = 41$**?** **No** ($z = 2y$ so it must be even)

CHALLENGES IN VALIDATING CLIENTS IN CRYPTOGRAPHIC PROTOCOLS (1)

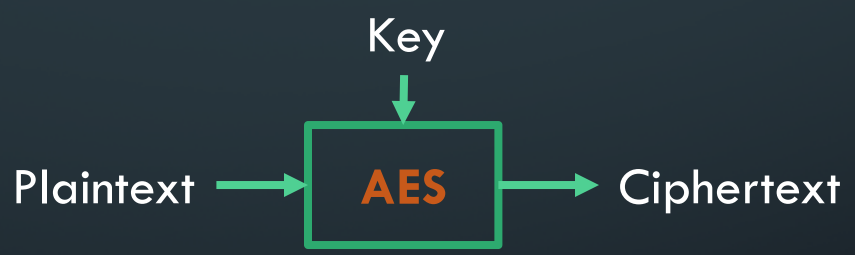
- Symbolic execution generally accommodates program variables with unknown values, but their *sizes* must be known
- Crypto protocols that hide sizes of client-side inputs (e.g., using padding) dramatically grow the search space



- Solution: Explore inputs of different sizes *in parallel*

CHALLENGES IN VALIDATING CLIENTS IN CRYPTOGRAPHIC PROTOCOLS (2)

- Some functions are too costly to execute on symbolic inputs
- Example: cryptographic functions
  - AES block cipher is a very complex formula of key and plaintext

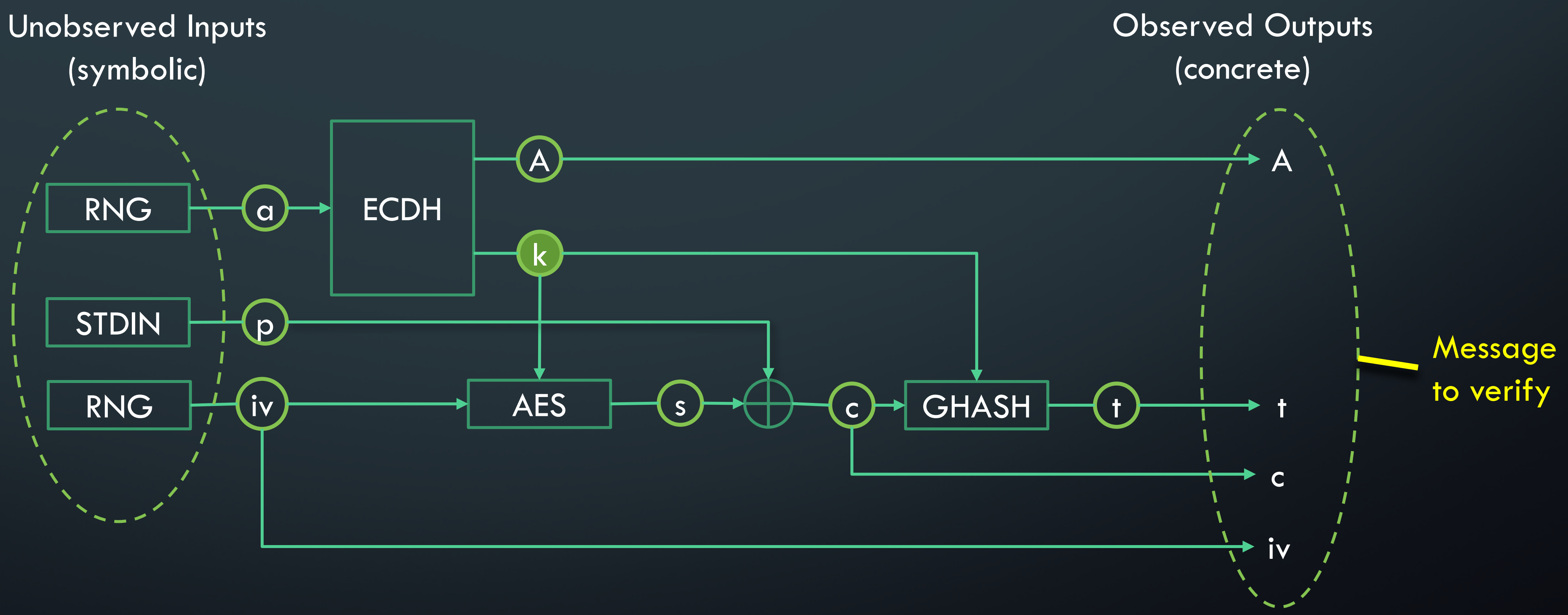


- Solution:
  - Give the verifier the session key
  - Defer executing prohibitive functions until inputs can be inferred
    - Any functions not executed then amount to assumptions

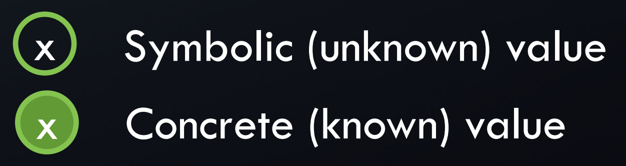
MULTIPASS SYMBOLIC EXECUTION

- Input: user specifies prohibitive functions, using an API
- Algorithm:
  1. Run symbolic execution.
     a) For each prohibitive function check if any inputs are symbolic
     b) If so, "skip" the function: return unconstrained symbolic output
     c) Otherwise, execute the function normally (all inputs are concrete)
  2. Concretize any variables with unique solution
  3. Repeat steps 1-2 until fixed point

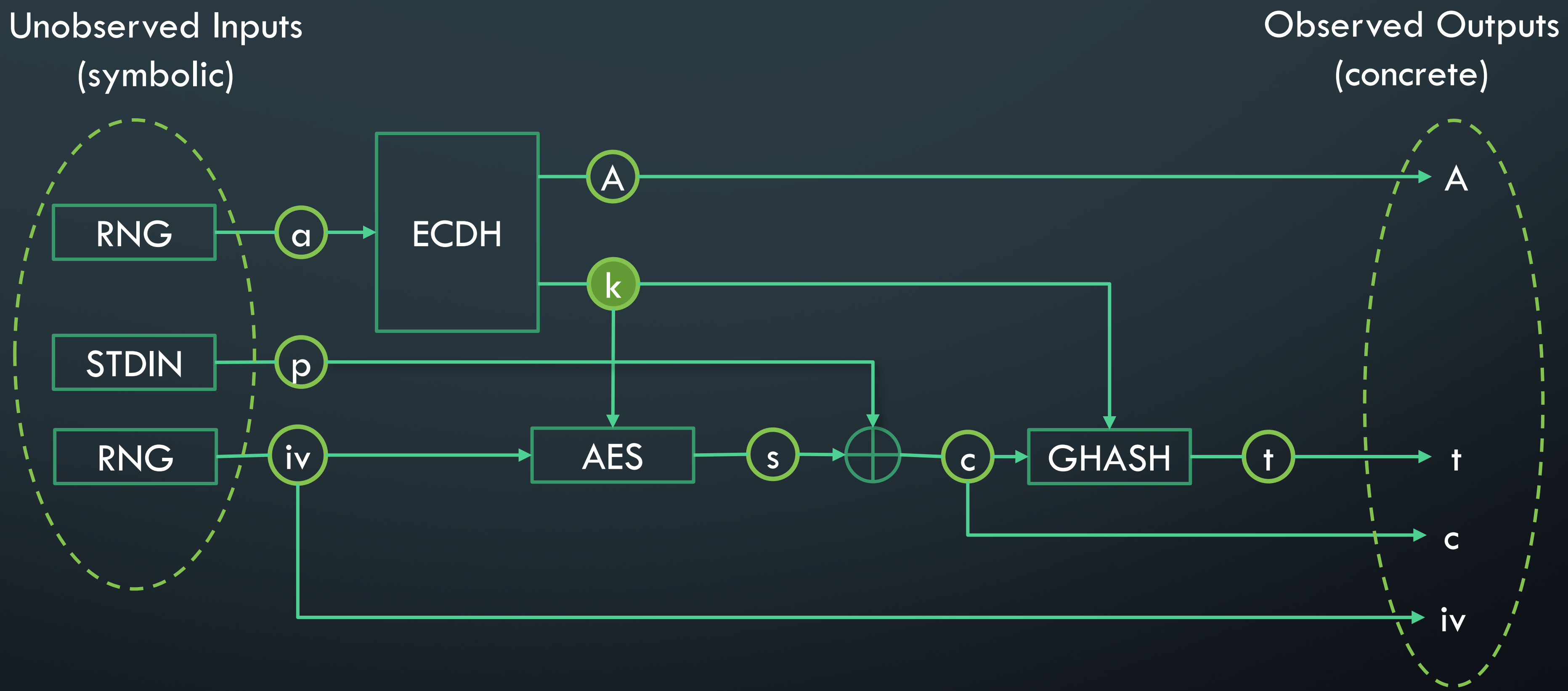
EXAMPLE: TLS CLIENT VALIDATION

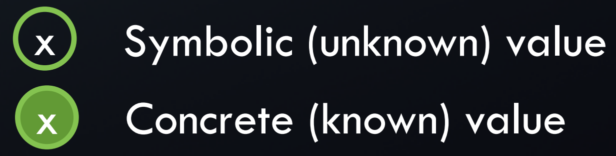


We assume knowledge of AES symmetric key, k, which is part of server state.

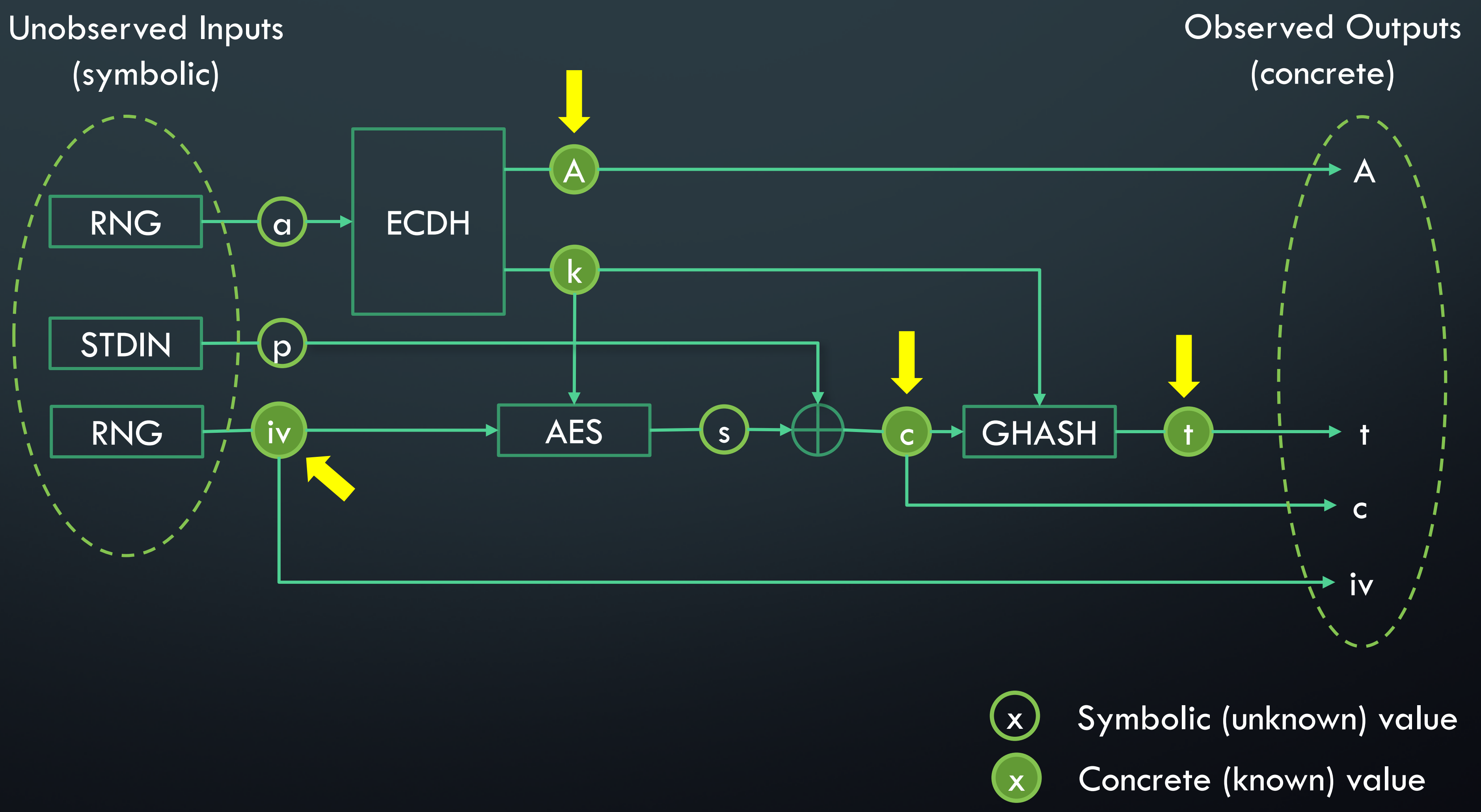


TLS CLIENT VALIDATION
PASS 1(A): SYMBOLIC EXECUTION

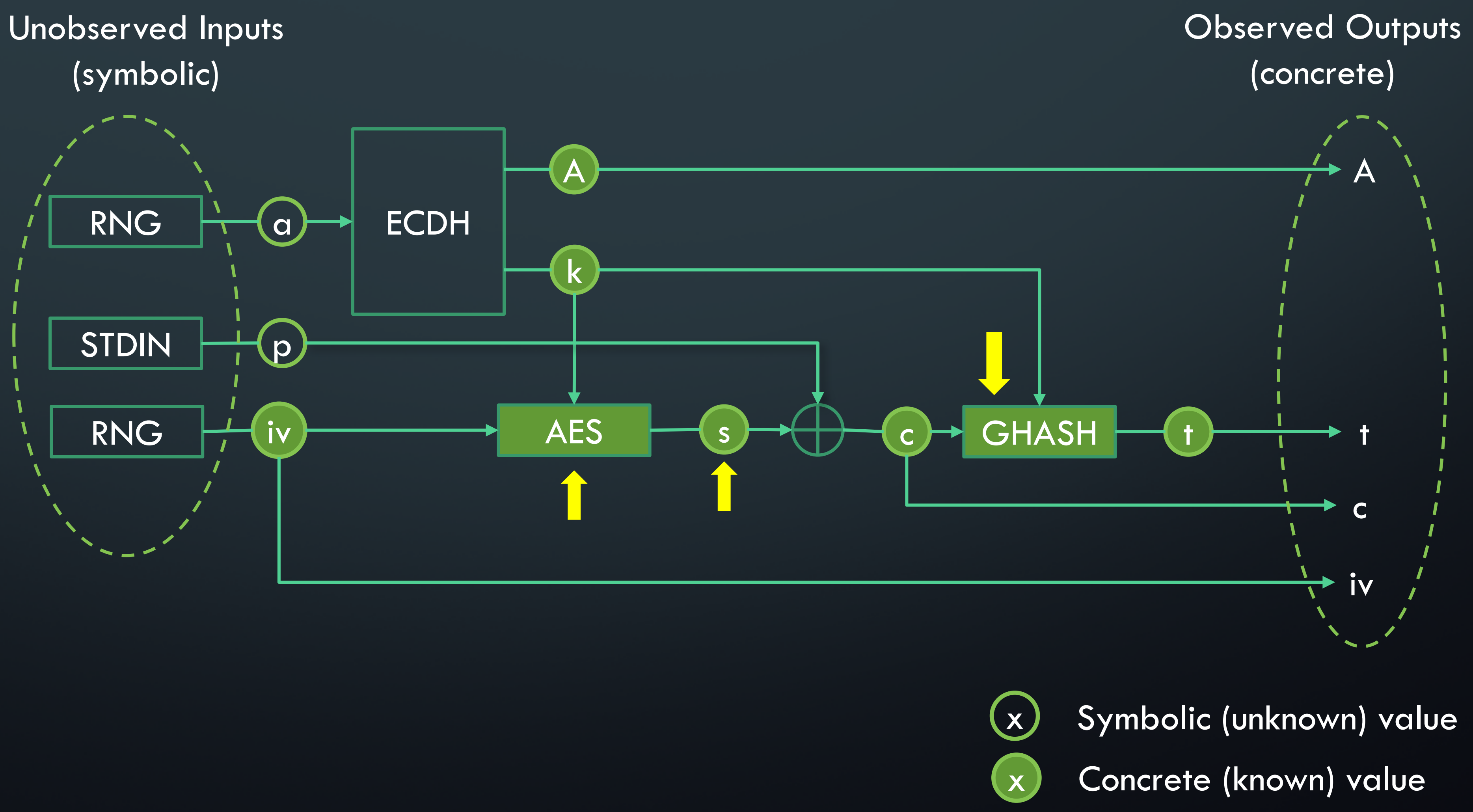




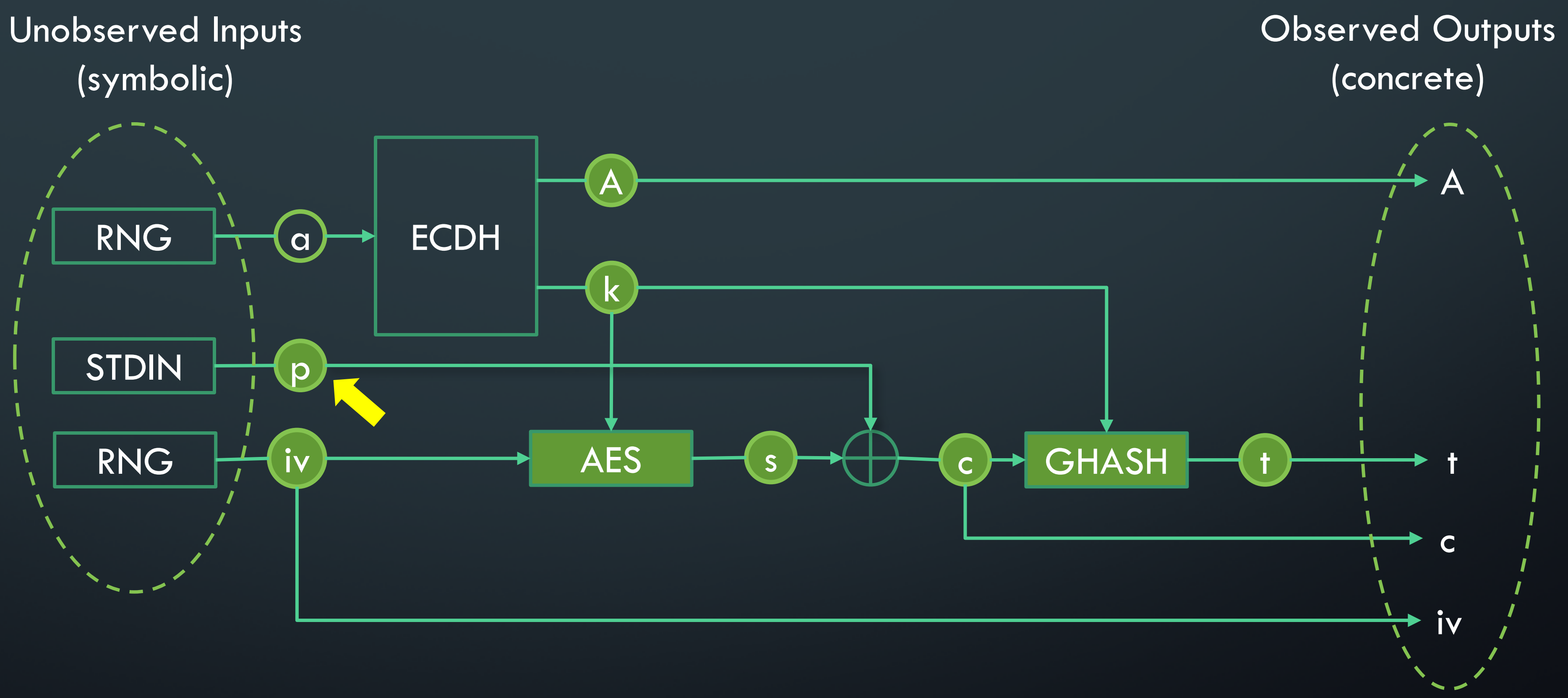
TLS CLIENT VALIDATION
PASS 1(B): CONCRETIZATION



TLS CLIENT VALIDATION
PASS 2(A): SYMBOLIC EXECUTION



TLS CLIENT VALIDATION
PASS 2(B): CONCRETIZATION

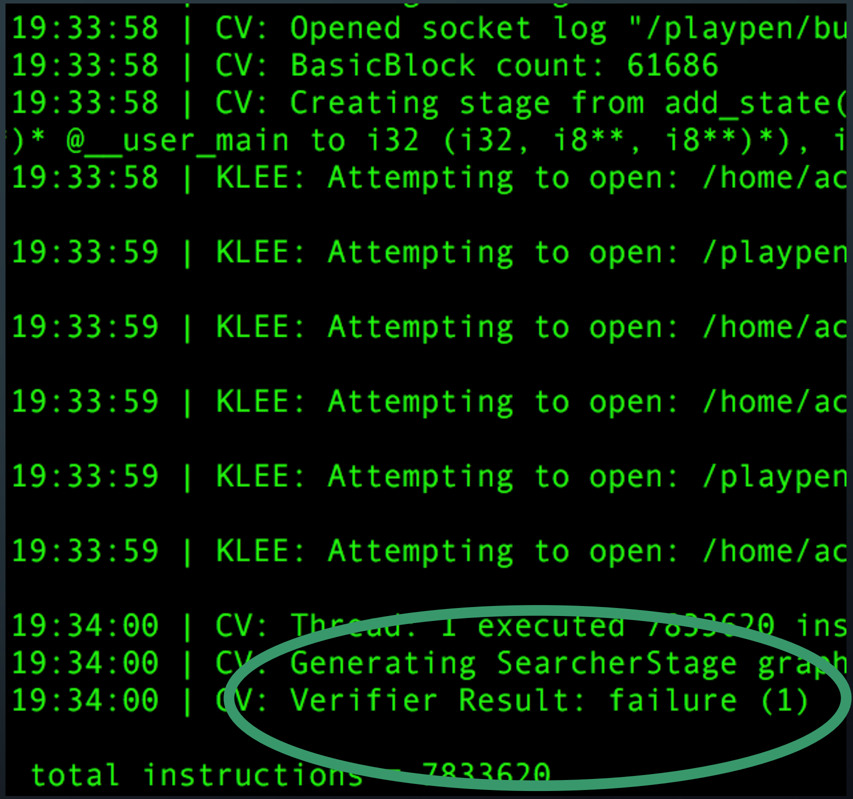


x Symbolic (unknown) value

x Concrete (known) value

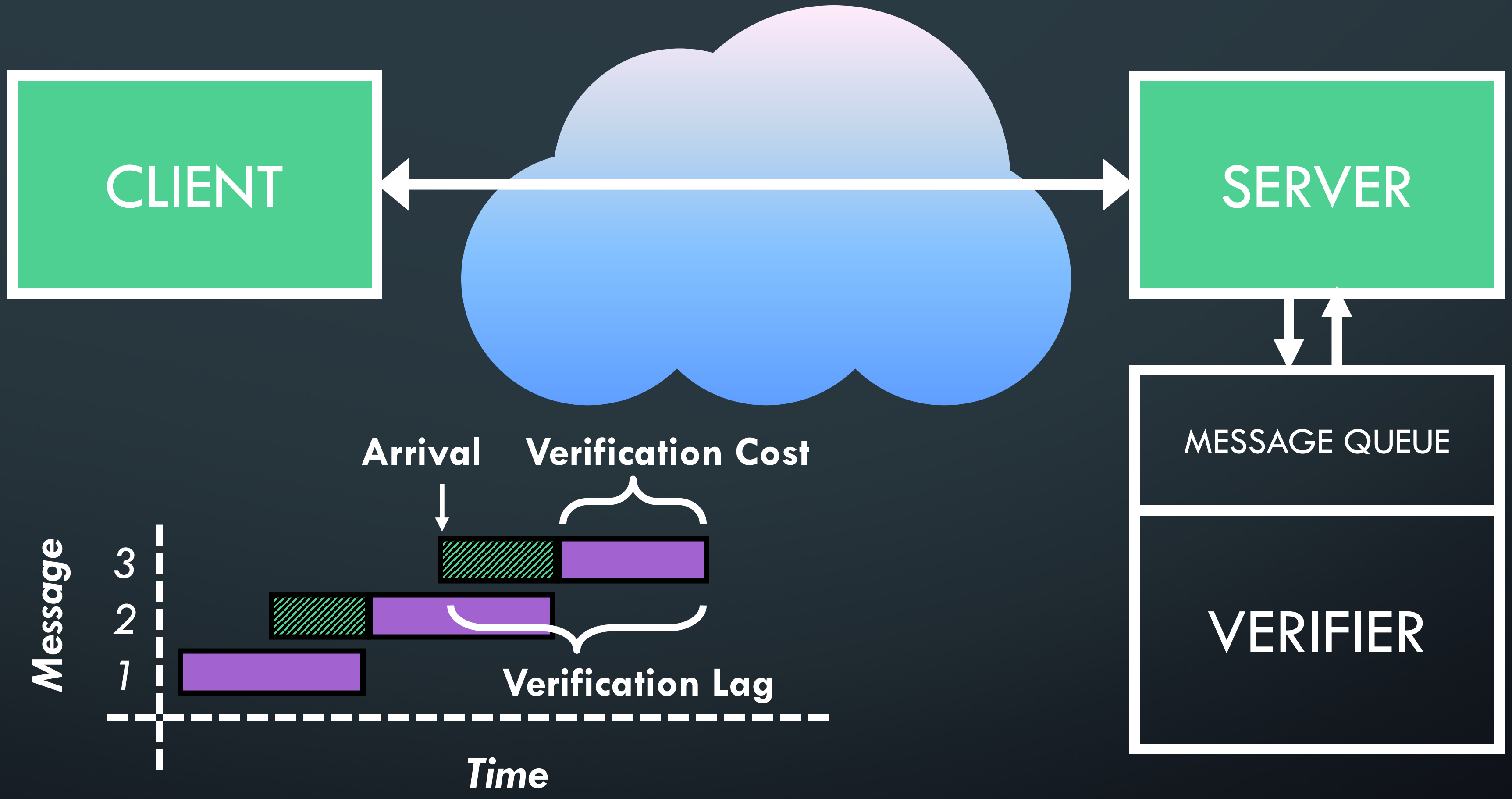
ASSESSMENT: DETECTING HEARTBLEED (WITHOUT LOOKING FOR IT)

- Malicious s_client
  - performs handshake
  - sends Heartbleed exploit
- Validation
  - Handshake is verified
  - No explanation found for malicious Heartbeat



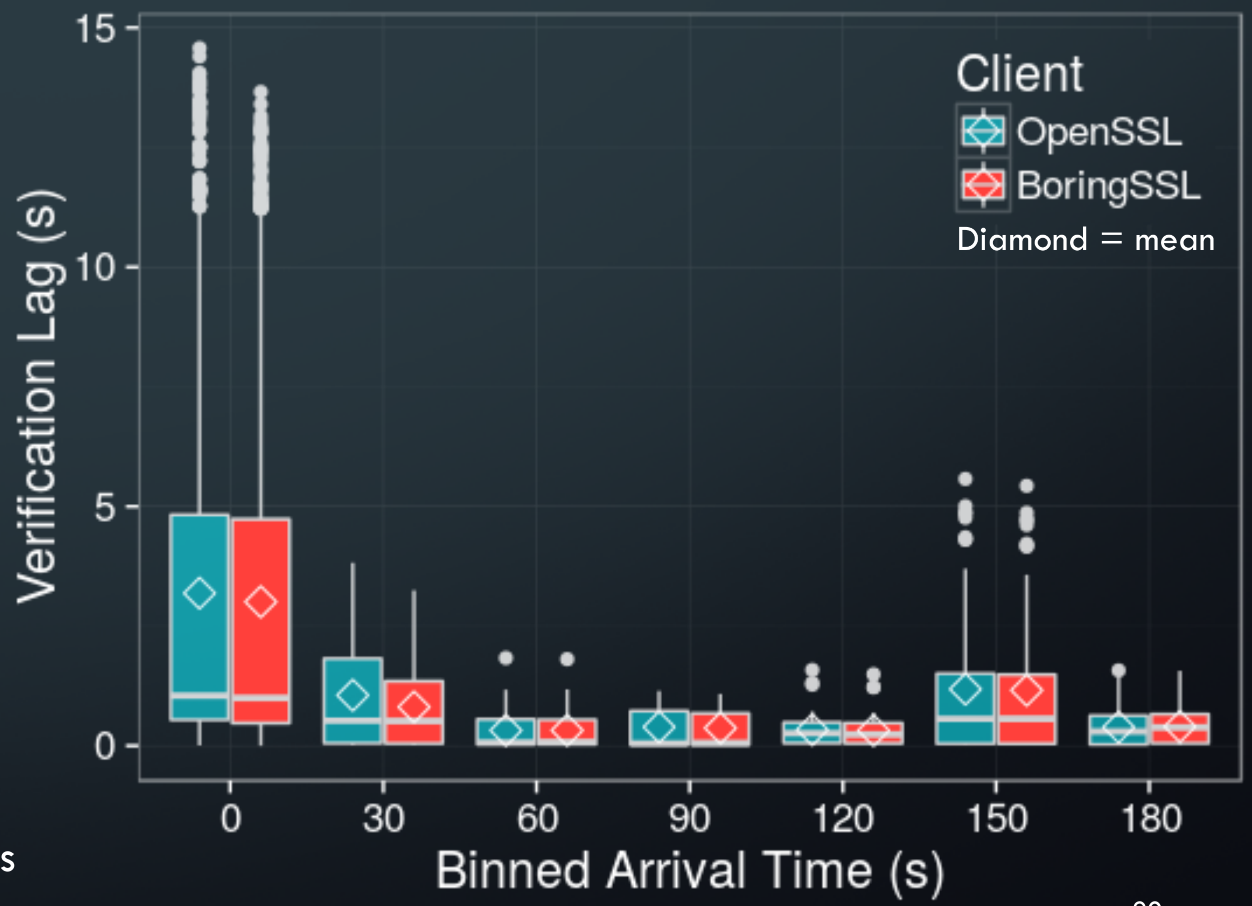
Detection in ~2s

MEASURING PERFORMANCE



PERFORMANCE EVALUATION

- 21 TLS 1.2 sessions from 3 min. of Gmail activity
- OpenSSL & BoringSSL command line clients
- Single-core verifier (3.2 GHz)
- Cost: 49ms per TLS record
- Lag: median 0.85s, max 15s



NOTE: without server-to-client appdata packets

OTHER EVALUATION MEASURES

- Parallelization / Stress Test
  - TLS 1.2 + up to 128 bytes of padding (from draft TLS 1.3)
  - 16-thread verifier keeps pace
- Invalid command attack: valid packets, illegal sequence
  - CVE-2015-0205 client authentication vulnerability
  - Verifier rejects attack traffic
- Confirm appropriateness of command line client
  - Unmodified Chrome browser interacting with Apache server
  - Verified using BoringSSL command line client

SUMMARY

- Behavioral verification for cryptographic clients
  - Multipass symbolic execution handles cryptographic functions
  - Parallelization optimizes search of large state spaces
- Detection of previously unknown client misbehavior
  - E.g., a Heartbleed exploit with no Heartbleed-specific configuration
- Performance roughly keeps pace with real workload
  - Behavioral verification on Gmail TLS sessions