A NAIVE IMPLEMENTATION OF BLINDBOX: PROTOCOL I

Deep Packet Inspection over Encrypted Traffic

Seyma Bodur, Fatma Demirtas, Cansin Yildiz

OUTLINE

- Introduction and Motivation
- BlindBox: Deep Packet Inspection over Encrypted Traffic
  - System Overview
- A Naive Implementation of BlindBox: Protocol I
  - System Overview
  - Demo
  - Limitations
- Questions?

INTRODUCTION AND MOTIVATION

WHAT IS DEEP PACKET INSPECTION (DPI)?

- In-network middleboxes use DPI to examine and alter packets
- Used to enforce security policies
    - Intrusion detection/prevention, exfiltration prevention, parental filtering etc.

DPI AND HTTPS

- HTTPS and other encryption protocols have dramatically grown in usage
- Packet payloads are encrypted, middleboxes can no longer inspect them
- To enable inspection, some systems support *insecure* HTTPS
- Main-in-the-middle attack on SSL

Functionality of Middleboxes

or

Privacy from Encryption

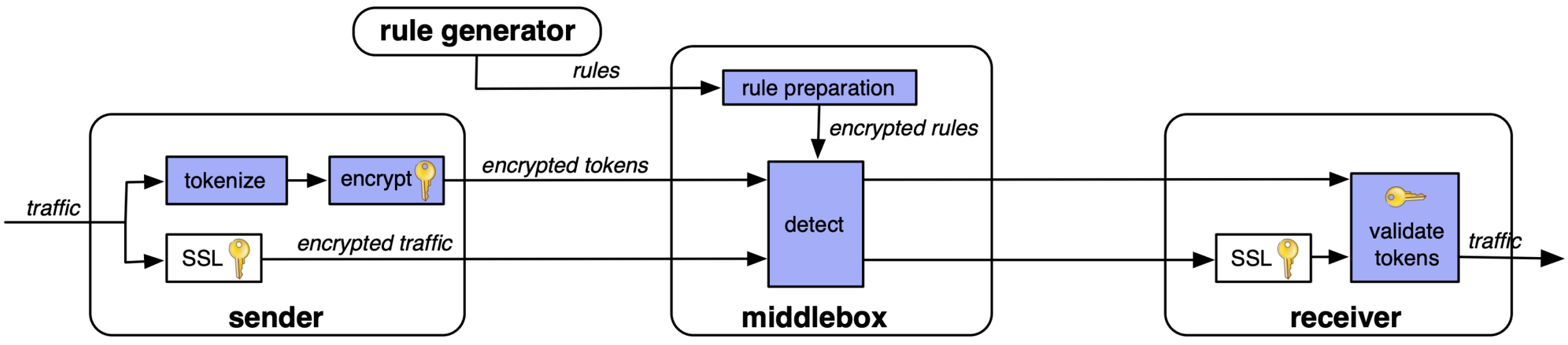
Can we get both?

BLINDBOX: DEEP PACKET INSPECTION OVER ENCRYPTED TRAFFIC

Justine Sherry, Chang Lan, Raluca Ada Popa, Sylvia Ratnasamy

CONNECTION SETUP

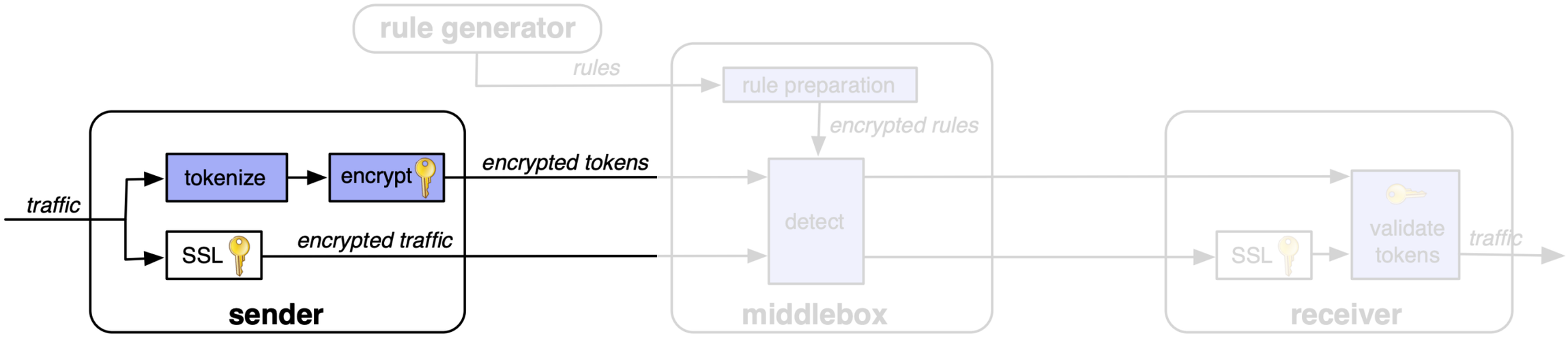
- At sender and receiver parts: $k_0 \rightarrow k_{SSL}, k_{rand}, k$
- At middlebox: obtaining rules from RG



SENDING TRAFFIC

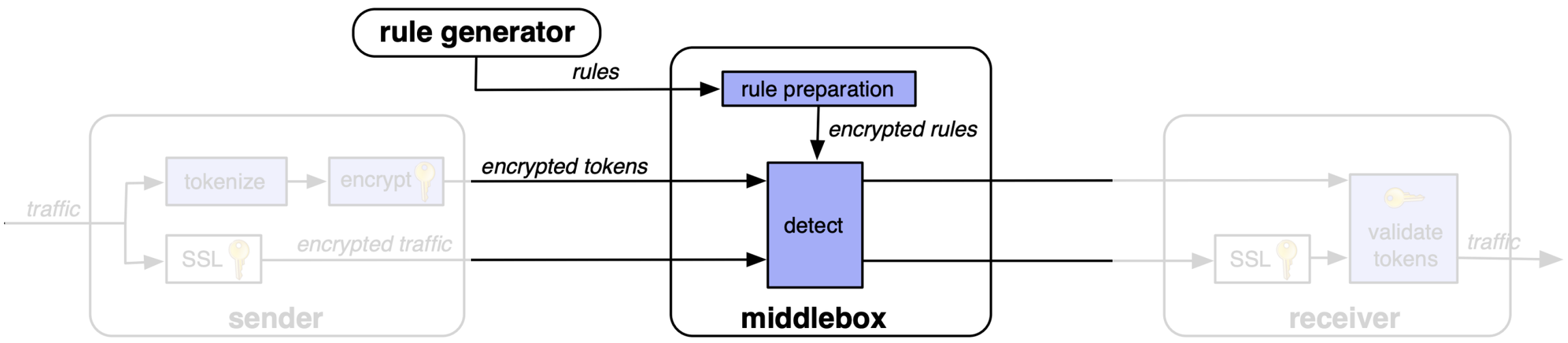
- Encryption of the traffic with SSL
- Tokenization
- Encryption of tokens

$$Enc_k(salt, t) = salt, AES_{AES_k(t)}(salt) \mod RS$$



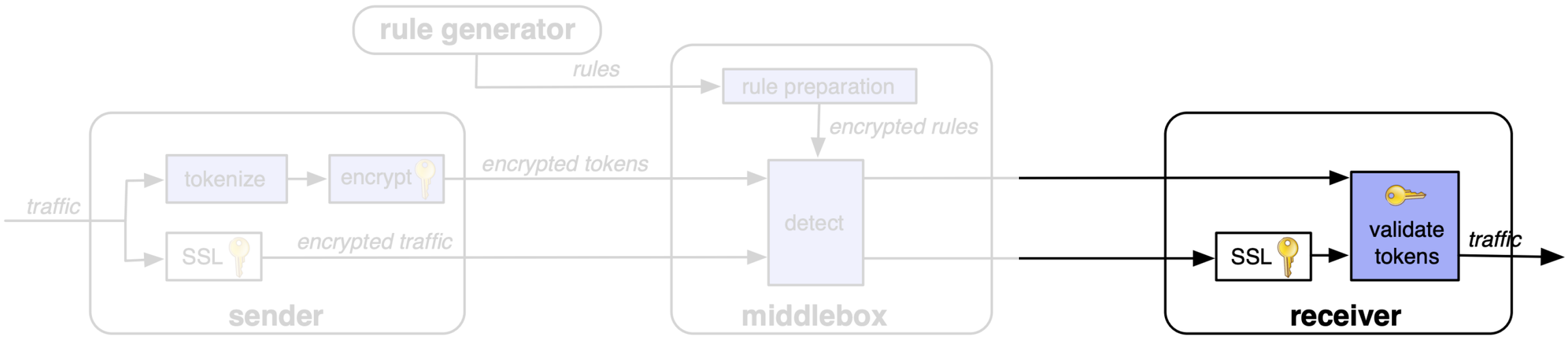
DETECTION

- Search for matching between encrypted rules and encrypted tokens



RECEIVING TRAFFIC

- Decrypting and authenticating the traffic with regular SSL.
- Checking tokens are encrypted properly by sender.

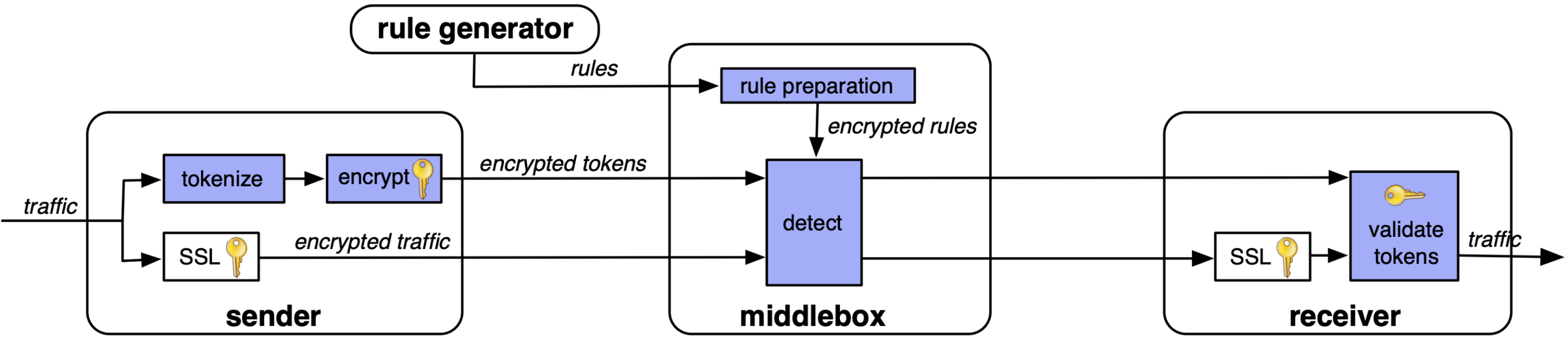


PROTOCOLS

- Protocol I: Basic Detection
- Protocol II: Limited IDS
- Protocol III: Full IDS with Probable Cause Privacy

A NAIVE IMPLEMENTATION OF BLINDBOX: PROTOCOL I

BLINDBOX: PROTOCOL I



SENDER.PY

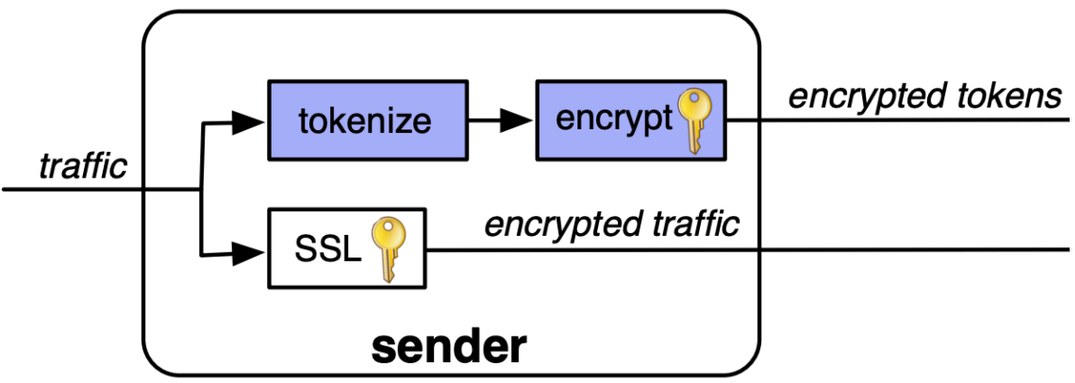
- Tokenization

```python
from Crypto.Cipher import AES
from Crypto.Hash import MD5

def encrypt(text):
    aes = AES.new(KEY, AES.MODE_CBC, INITIAL_VALUE)
    padding_threshold = int(math.ceil(len(text) / 16.0) * 16)
    return aes.encrypt(text.ljust(padding_threshold))

def token(text):
    h = MD5.new()
    h.update(encrypt(text))
    return h.digest()


def tokenize(input):
    tokens = []
    for i in range(len(input)):
        tokens.append(token(input[1][i:i + 8]))
    return tokens
```



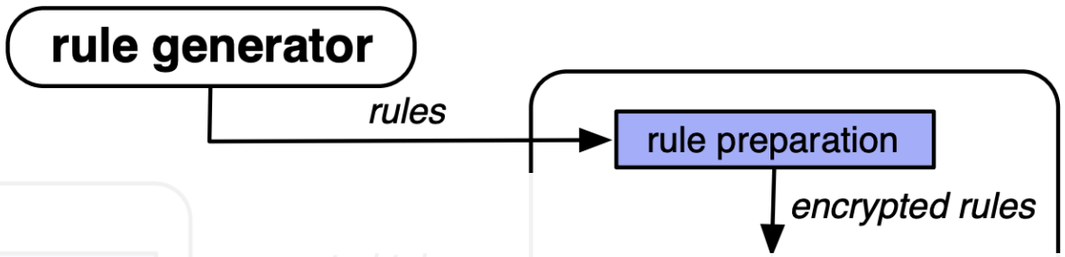
- Layer Definition

```python
from scapy.all import Packet, StrFixedLenField, X3BytesField

TYPE_BLINDBOX = 0x811ad8

class BlindBox(Packet):
    name = "BlindBox Packet"
    fields_desc = [
        X3BytesField("protocol", TYPE_BLINDBOX),
        StrFixedLenField("token", "                ", 16)
    ]
```

RULE_GENERATOR.PY



- Rule Generation

```
from Crypto.Cipher import AES
from Crypto.Hash import MD5

def encrypt(text):
    aes = AES.new(KEY, AES.MODE_CBC, INITIAL_VALUE)
    padding_threshold = int(math.ceil(len(text) / 16.0) * 16)
    return aes.encrypt(text.ljust(padding_threshold))

def token(text):
    h = MD5.new()
    h.update(encrypt(text))
    return h.digest()
```

MIDDLEBOX.P4

- Parser

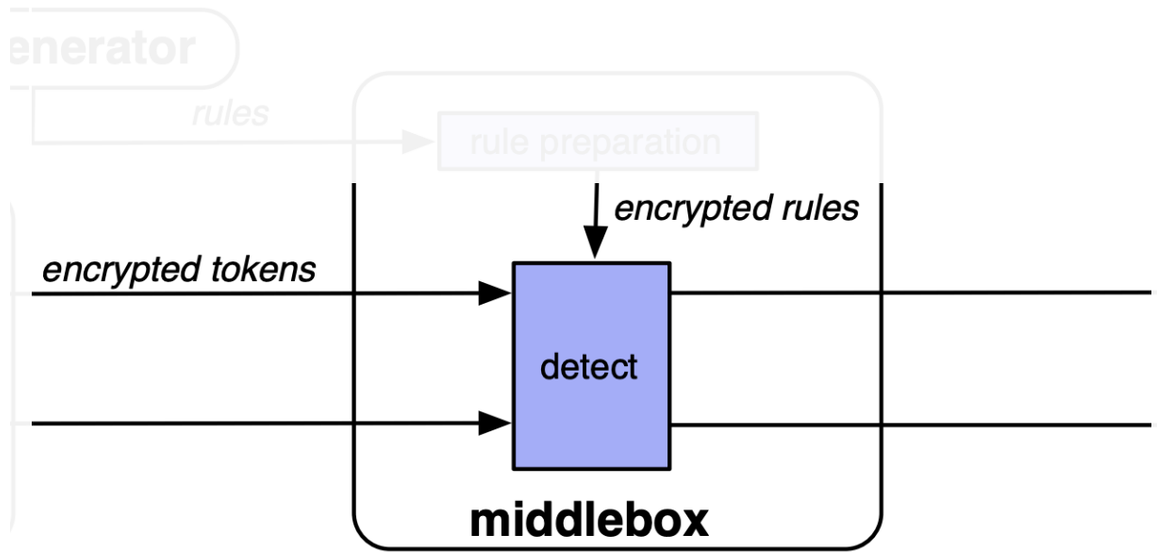
```p4
const bit<24> TYPE_BLINDBOX = 0x811ad8;

header tcp_t {
  // Other standard TCP headers.
  // .
  // .
  bit<24> protocol;
}

header blindbox_t {
  bit<128> token;
}

parser MyParser(packet_in packet, out headers hdr) {
  // Other standard states needed to parse Ethernet, and IP headers.
  // .
  // .
  state parse_tcp {
    packet.extract(hdr.tcp);
      transition select(hdr.tcp.protocol) {
        TYPE_BLINDBOX: parse_blindbox;
        default: accept;
      }
  }

  state parse_blindbox {
    packet.extract(hdr.blindbox);
    transition accept;
  }
}
```



- Ingress

```p4
control MyIngress(inout headers hdr) {
  // Other standard actions needed for a proper Ethernet, IP, and TCP handling.
  // .
  // .
  apply {
    if (hdr.ipv4.isValid()){
      if (
        hdr.tcp.isValid() && hdr.blindbox.isValid() &&
        (
          hdr.blindbox.token == 128w0x52a5671d0308d078677f22f6f824a4b2 ||
          hdr.blindbox.token == 128w0x280f0fdc5e06531f67e5fb32bceb7ee1
        )
      ) {
        drop();
        return;
      }
      ipv4_lpm.apply();
    }
  }
}
```

RECEIVER.PY

- Receiving Tokens

```
class BlindBoxSession:
    def __init__(self):
        self.received_tokens = []

    def add_token(self, token):
        self.received_tokens.append(token)


session = BlindBoxSession()

def handle_pkt(pkt):
    global session
    if TCP in pkt:
        if ('\x00' + bytes(pkt[TCP].payload)[:3]) == struct.pack(">L", TYPE_BLINDBOX)
    :
            token = str(pkt[TCP].payload)[3:]
            session.add_token(token)
        else:
            payload = decrypt(str(pkt[TCP].payload))
            session = BlindBoxSession()
```


- Validating Tokens

```
    def validate(self, payload):
        print "Validating session"
        for i in range(len(payload)):
            self.generated_tokens.append(token(payload[i:i + 8]))

        self.is_valid = self.received_tokens == self.generated_tokens
```

```
# At ‘‘Node: h1’’ Xterm window
root@p4:~/code/ceng781-tp# python -m client.sender "This is a safe message."
['/home/p4/code/ceng781-tp/client/sender.py', 'This is a safe message.']
Sending on interface h1-eth0 to 10.0.2.2

# At ‘‘Node: h2’’ Xterm window
root@p4:~/code/ceng781-tp# python -m client.receiver
Sniffing on h2-eth0
Got a BlindBox packet with token 128w0x6707768d0faea83ec989f79017551413
Got a BlindBox packet with token 128w0xea8a4b9caf8100dc0bcf4639719ae15e
Got a BlindBox packet with token 128w0x49da97486c8303497a95941b1a36b627
Got a BlindBox packet with token 128w0x36e08cc0254802587c39e397b720a8a3
Got a BlindBox packet with token 128w0xcab941fecd1efd7c5828d9a94ffceabd
Got a BlindBox packet with token 128w0x6d4d23d36fd48ad9fa5be9d7857c2608
Got a BlindBox packet with token 128w0xf28432026b88e5579bfc0320f1185033
Got a BlindBox packet with token 128w0xdcdb0d2396060b487d78bde16f41a0cd
Got a BlindBox packet with token 128w0xbd[illegible]072ec812521196d37acf7
Got a BlindBox packet with token 128w0x583962e00[illegible]4134dd8f175ab444fe1ed
Got a BlindBox packet with token 128w0x334e8ce53441846f986750b2bb29c1b8
Got a BlindBox packet with token 128w0xdfdbaf6853ccda11c1973d736fdc124f
Got a BlindBox packet with token 128w0x997d0f2fafcda834f6af9df7f4824594
Got a BlindBox packet with token 128w0xb5da372f43d728a1f0556ffe6cd3588a
Got a BlindBox packet with token 128w0x5d77812e0c5ac2fd8307a17f46c247f8
Got a BlindBox packet with token 128w0x3c35d138ea44822501c97f6f99835c29
Got a BlindBox packet with token 128w0xfd142756822a172588fe019aa65a6358
Got a BlindBox packet with token 128w0x3f7611beb2c3196830b13efecbf00572
Got a BlindBox packet with token 128w0xc834923b0417a20aeef73a924c1f00ef
Got a BlindBox packet with token 128w0x939c6cfe9da1eda33157c0e0f407df09
Got a BlindBox packet with token 128w0xb3284e55dda118060437f7e5ff9d1181
Got a BlindBox packet with token 128w0xf5cead6cf416987c5922cb82eccf938a
Got a BlindBox packet with token 128w0x042d0b3c5b069d9a1d9b4ecd58733d7d
Validating session
Got a VALID TCP packet with payload "This is a safe message."
```

DEMO

LIMITATIONS

- Rule detection is limited to 8 to 15 bytes only
  (vs. BlindBox claims to detect >8 bytes)

- Encrypted token values are hardcoded in P4

- A single token packet uses 19 bytes
  (vs. BlindBox uses 5 bytes per token packet)

- SSL is not implemented.
  Relying on AES for the actual traffic instead.

QUESTIONS AND COMMENTS?

Thank you.