# A Security Model and Fully Verified Implementation for the IETF QUIC Record Layer

Presented by: Jake Ginesin

# Background 1: What is a Refinement Type?

- They're just types with predicates attached
- The predicates have no quantifiers ($\forall$, $\exists$) to ensure type checking is decidable
- SMT solvers (Z3, cvc5) are used for verification

```
type nat = int[v|0 <= v]
```

```
val size : x:array(α) ⇒ nat[v|v = length(x)]
val get  : x:array(α) ⇒ nat[v|v < length(x)] ⇒ α
```

Src: Refinement Types: A Tutorial

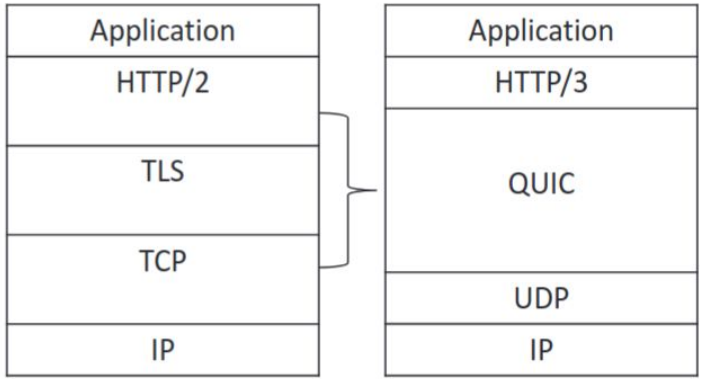
# Background 2: What is F* and F#?

- F#: A functional programming language
- F*: A language with refinement types. *Extracts* to OCaml or F#

Q1: Can we trust the extractions? Recall ironfleet had the verified transformations

# Background 3: What is QUIC?


- (Relevant) differences from TCP/TLS:
  - QUIC is over UDP
  - QUIC combines the TCP/TLS handshakes
  - allows for 0-RTT (w/o forward secrecy) and 1-RTT



# Motivations


- The IETF and Google’s QUIC drafts differ significantly
- IETF’s QUIC does *not* inherit the security props of TLS 1.3
- Modern IETF QUIC spec is not well analyzed

# Goals of this paper

- Verify the *Record Layer* (i.e., packet formatting and encryption)
- A security model for the record layer
- Formal specs
- A provably safe record layer & implementation

Q2: Has there been explicit verification of the protocol logic? I'd imagine that's easier.
Q3: What's even the utility of a F# implementation?

# QUIC details covered (background-ish)

They authors selectively introduce parts of QUIC that they verify(?):

- TLS handshake interface
- Connection establishment
- Packet headers
- Connection identifiers
- Packet numbers & stream encryption
- Transport parameter negotiation, version negotiation

Q4: Are they leaving anything out? I am not a QUIC expert D:

# QUIC Packet Encryption (QPE) intro !

Payload:

Uses AEAD negotiated with TLS, which assures *confidentiality* and *authenticity*

Header:

A new “rather convoluted” protection scheme…

They provide a *super detailed* construction

# QUIC Packet Encryption (QPE) defs

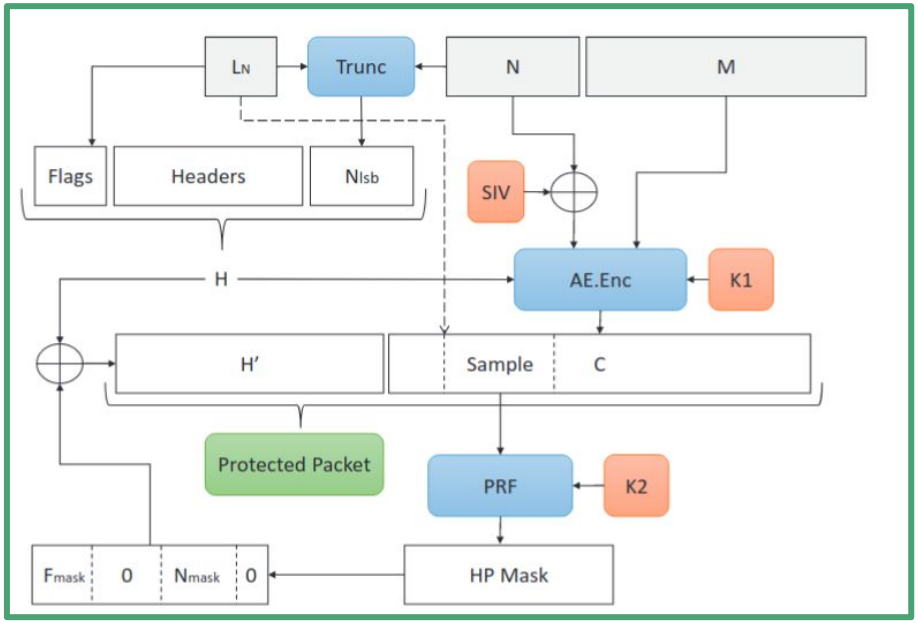
- **QPE[AE,F]**: given by **AE**AD and pseudo-random func **F**

Defines the following…

- Keygen
- Decode/Encode
- Encrypt/Decrypt
- PNenc/PNdec

Q5: What is PNenc/dec?

# QUIC Packet Encryption Diagram




Legend:

- **N**: packet num (**Nlsb**: truncated)
- **M**: plaintext msg
- **Ln**: “least-significant” bytes of N”
- **PRF:** pseudo-random function
- **SIV**: 12 bytes of static “IV”
- **C**: Cybertext
- **H**: Header (**H’** is protected)
- **K1,K2** -> Secrets for AEAD

Q6: how are least significant bytes chosen?!

# Reductions via Games: Hardness Assumption

**Game** $\mathsf{PRF}^b(\mathsf{F})$

$T \leftarrow \varnothing$

$k \xleftarrow{\$} \{0,1\}^{\mathsf{F}.\ell_K}$

**Oracle** $\mathsf{Compute}(X)$

**if** $b = 1$ **then**

  **if** $T[X] = \perp$ **then** $T[X] \xleftarrow{\$} \{0,1\}^{\mathsf{F}.\ell}$

  **return** $T[X]$

**return** $\mathsf{F}.\mathsf{compute}(k, X)$

- Adversary is given oracle *Compute(X)*
- Oracle behaves if b=0 (random **F** is used) or b=1 (random values given instead)
- F is secure iff adversary cannot determine if (b=0) or (b=1)

# Packet Encryption Initial Security Definition?

**Game** $\mathsf{AE1}^b(\mathsf{SE1})$

$T \leftarrow \varnothing;\ k \xleftarrow{\$} \mathsf{SE1.gen}()$

**Oracle** $\mathsf{Decrypt}(N, C, H)$

**if** $b = 1$ **then**

  $M \leftarrow T[N, C, H]$

**else**

  $M \leftarrow \mathsf{SE1.dec}(k, N, C, H)$

**return** $M$

**Oracle** $\mathsf{Encrypt}(N, M, H)$

**assert** $T[N, \_, \_] = \perp$

**if** $b = 1$ **then**

  $C \xleftarrow{\$} \{0,1\}^{|M| + \mathsf{SE1}.\ell_{\mathrm{tag}}}$

  $T[N, C, H] \leftarrow M$

**else**

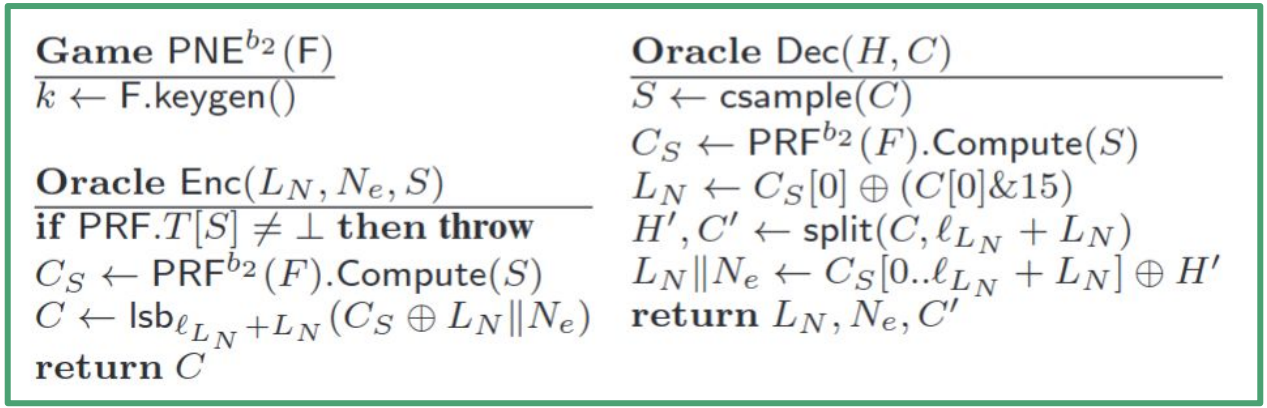
  $C \leftarrow \mathsf{SE1.enc}(k, N, M, H)$

**return** $C$

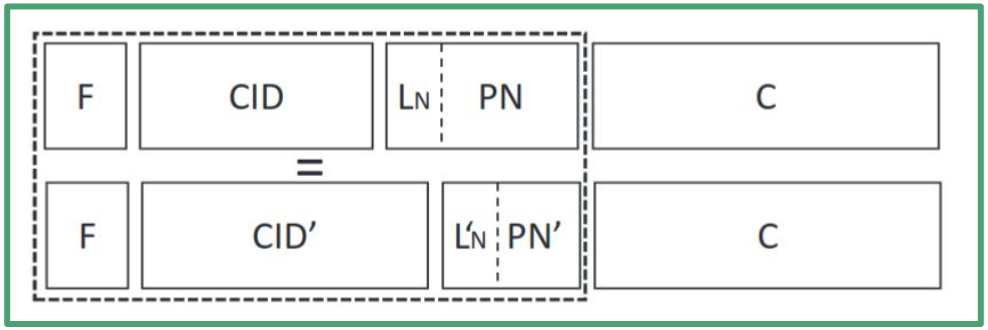
Can the adversary discover the nonce (one-time value) to decrypt a given ciphertext in polytime?

# Nonce Confidentiality Security Proof

- Long story short, we *reduce* the hardest of this game to the hardness of the assumptions:



# An Attack: Connection ID Authentication




- Modifying ***least significant bytes*** (Ln) => get peers to disagree on ***destination connection ID*** (CID) end location
- Attacker easily “wins” game via described routine
- Submitted to the IETF. They didn’t fix it. (HN2 proposed)

Q7: Should the connection ID be protected by TLS?

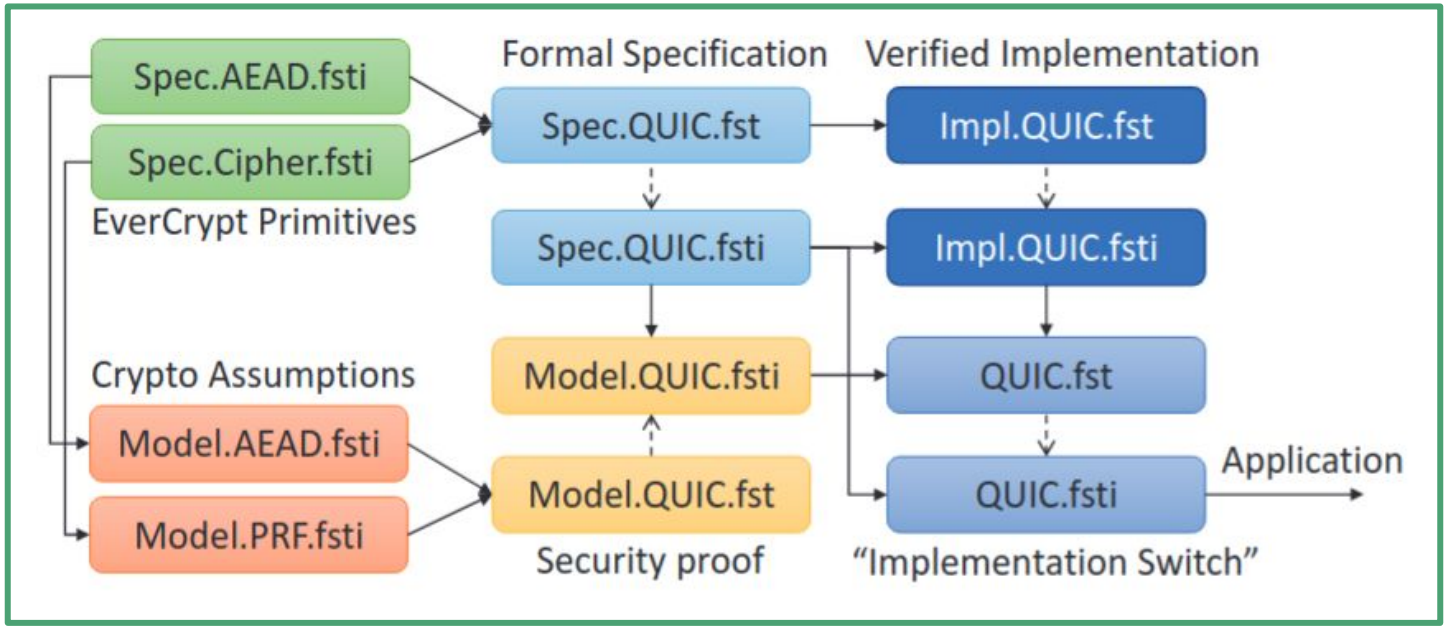
# Reference Implementation Overview

- Record Layer in **F***
- Verified for various cryptographic properties
- Proofs discharged with Z3
- Important part

- Protocol Logic in **Dafny**
- Verified for memory safety, type safety, termination, integer overflows
- Proofs discharged with Z3
- Extended Dafny to add C++ backend support!?

Q8: where is the extended dafny ver?!

# Record Layer Structure



# Record Layer Prop #1

```
val lemma_header_parsing_correct: ... → Lemma
  (parse_header cid_len last
    (format_header (append h c)) = Success h c))
```

- Prop #1: The correctness of the packet-number decoding
- The parse_header inverts the format_header

# Record Layer Prop #2

```
val lemma_header_parsing_safe: ... → Lemma (requires ... ∧
    parse_header cid_len last b1 = parse_header cid_len last b2)
  (ensures parse_header cid_len last b1 = Failure ∨ b1 = b2)
```

- Prop #2: Correctness and injectiveness of the header parsing

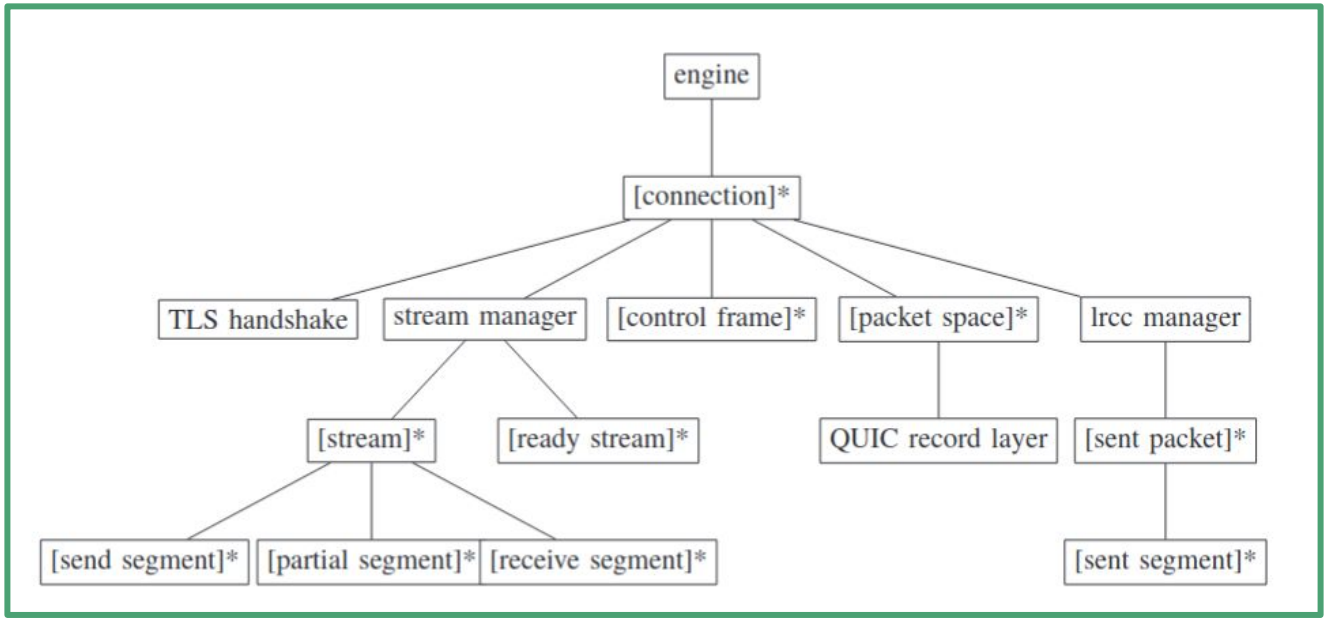
# Record Layer Prop #3

```
val lemma_encrypt_correct:
  a:ea (* the AE algorithm negotiated by TLS *) →
  k:lbytes (ae_keysize a) (* the main key for this AE algorithm *) →
  siv: lbytes 12 (* a static IV for this AE algorithm *) →
  hpk: lbytes (ae_keysize a) (* the header key for this AE algorithm *) →
  h: header  (* Note the condition on the CID length below *) →
  cid_len: nat {cid_len ≤ 20 ∧ (MShort? h ⇒ cid_len = dcid_len h)} →
  last: nat{last+1 < pow2 62 } →
  p: pbytes' (is_retry h) { has_payload_length h ⇒
     U64.v (payload_length h) == length p + AEAD.tag_length a } →
  Lemma (requires is_retry h ∨ in_window (pn_length h − 1) last (pn h))
(ensures decrypt a k siv hpk last cid_len
   (encrypt a k siv hpk h p) = OK h p)
```

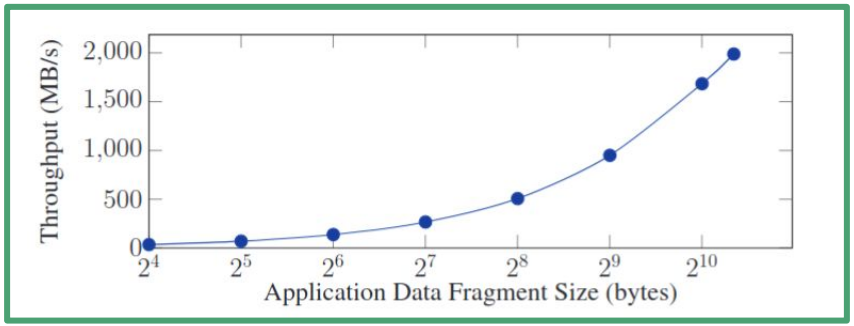
- Prop #3: correctness of header and payload decryption
- Simply, encryption inverts decryption

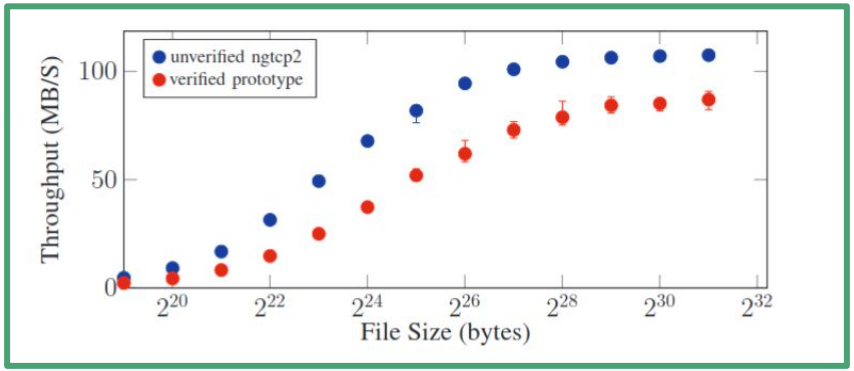
Q9: any other props?

# Protocol Logic Structure (Dafny)



# Implementation Throughput Details



# Summary

| Modules | LoC | Verif. | C/C++ LoC |
|---|---|---|---|
| Verified Record Layer (§IV) | | | |
| QUIC.Spec.* | 5,463 | 5m12s | - |
| QUIC.Impl.* | 5,509 | 6m32s | 4,640 |
| QUIC.Model.* | 1,751 | 3m12s | - |
| LowParse.Bitfields.* | 2,011 | 1m29s | - |
| LowParse.Bitsum.* | 2,502 | 2m05s | - |
| Total | 9,836 | 16m30s | - |
| QUIC Reference Implementation (§V) | | | |
| Connection mgmt | 4,653 | 14m12s | - |
| Data Structures | 651 | 9s | - |
| Frame mgmt | 1,990 | 1m50s | - |
| LR & CC | 758 | 11s | - |
| Stream mgmt | 1,495 | 3m25s | - |
| Misc | 118 | 2s | - |
| FFI | 558 | 9s | 1461 |
| Server & Client | - | - | 648 |
| Total | 10,223 | 19m46s | 2,109 |

# Any questions? :D