

DEFJAM to CUTLASS One Year After

Todd MacDermid

Jack Lloyd

Kathy Wang

(John Schweitzer)

Quick Recap

- Founded at Notacon
- DEFJAM last year
- Talked design



CUTLASS Design Goals

- Easy enough to use for broad adoption
- Cross-Platform - Avoid tweaky APIs
- Secure by default, encrypted channels, resistant to traffic analysis
- Useful with small network effect
- Extendable (both functionality & paranoia)
- Not dependent on central servers

CUTLASS Anti-Goals

- Not a strong anonymity system
- Not restricted to existing standard protocols
- Useful without complete meshed

What's Done?

- Audio
- Key exchange
- Housekeeping
- Text messages
- File transfer
- Directory support in library

Demo



Why Are You Here?

- We need help **OBVIOUS**
 - Coding, testing, design suggestions
- We've got info you'd need to know
- Not tied to slides, questions anytime

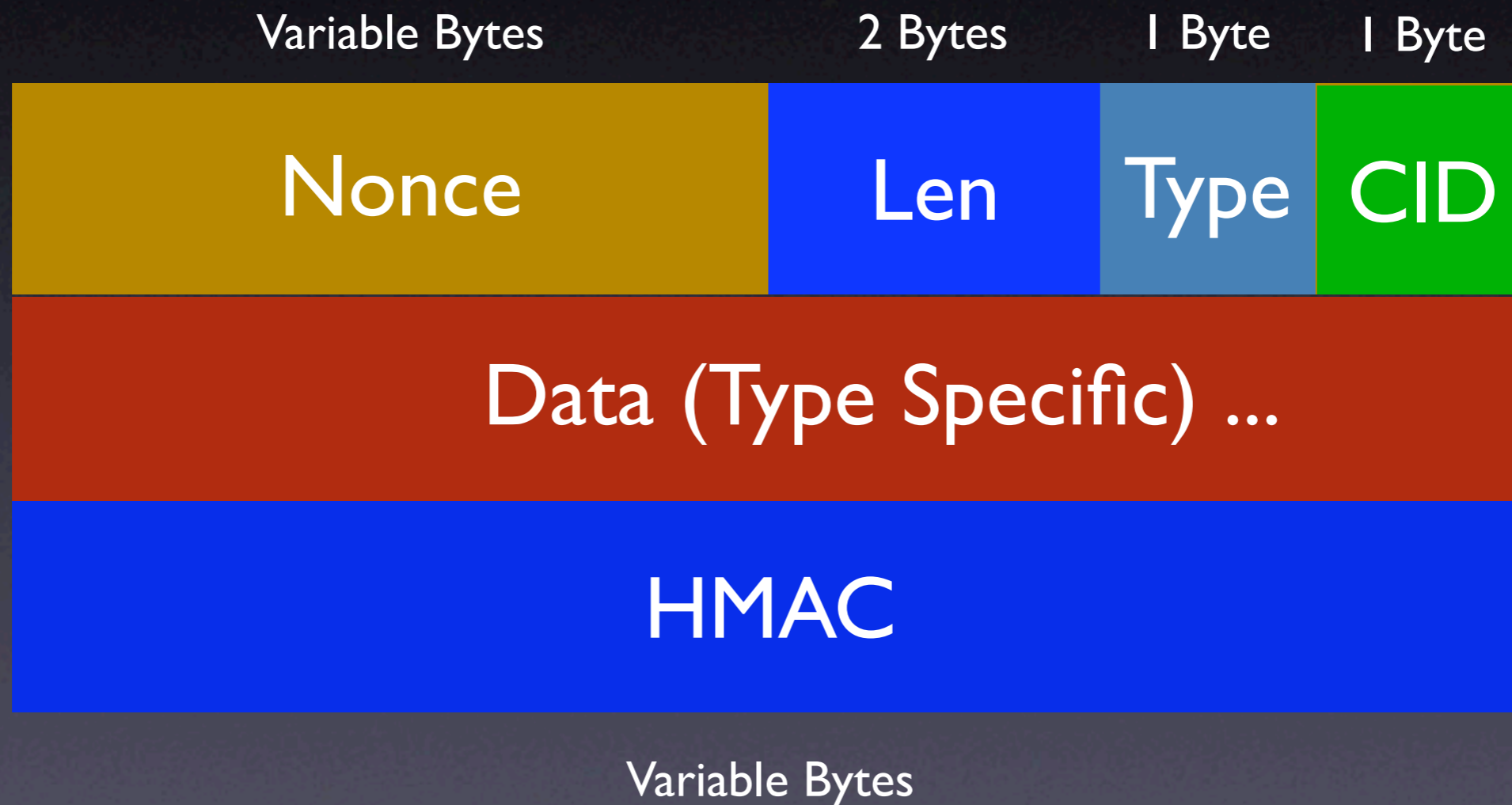
What Does the Material Cover?

- Protocol
- Cryptography
- Internal structure and flow
- API
- Test suite
- Documentation

CUTLASS Protocol Overview

- Default of UDP for all traffic
- Allows anonymizing and traffic analysis defeating measures, but not enabled by default
- Allows individual and group messages/transfers
- Allows sound, file, and text transfers

CUTLASS Packet Structure



CUTLASS Packet Types

- Key Exchange
- Ping/Pong
- Connection Information Req/Resp
- Audio
- Reliable Transport

Connection Information Packet

4 Bytes

Variable Bytes

Capability Flags

Nickname

CUTLASS Audio

4 Bytes

Variable Bytes

Sequence #

Speex Payload

CUTLASS Reliable Transport

- Transport Packet Types
 - Init
 - Init Ack
 - Send
 - Ack (Request)
 - Channel Reset

Reliable Transport Types

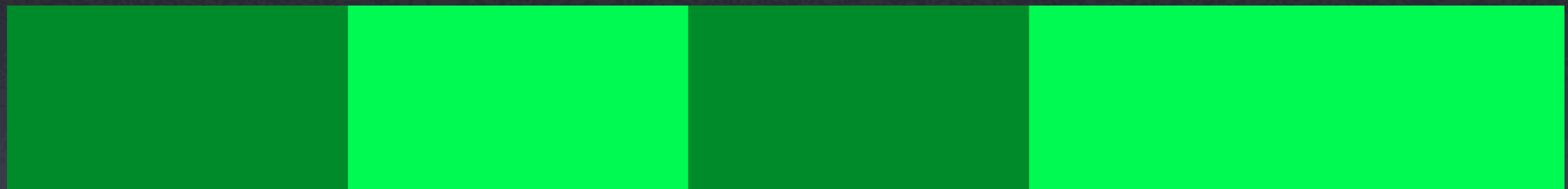
- Memory Backed
 - Messages
 - Directory Requests
- File Backed
 - Files

CUTLASS Transport Layer

“Gap”-based requests

0

4500



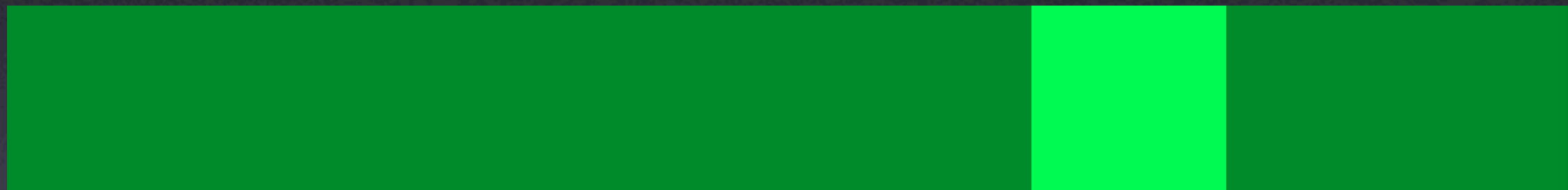
Request: 0-4500

CUTLASS Transport Layer

“Gap”-based requests

0

4500



Request: 1000-2000,3000-4500

CUTLASS Transport Layer

“Gap”-based requests

0

4500



Request: 3500-4000

CUTLASS Transport Rate-Limiting

- Requests immediately get one response
- Successful request/response pair increases unsolicited rate by one PPS
- Periodically send unsolicited data according to rate
- If number of gaps increases, decrease unsolicited rate

CUTLASS Transport Stats

Copying 34 MB file over 10Mbps local link:

- SCP: 45 seconds
- CUTLASS: 53 seconds

Simultaneous copy bandwidth consumption:

- 75% of bandwidth used by SCP
- 25% of bandwidth used by CUTLASS

CUTLASS Transport Layer Advantages

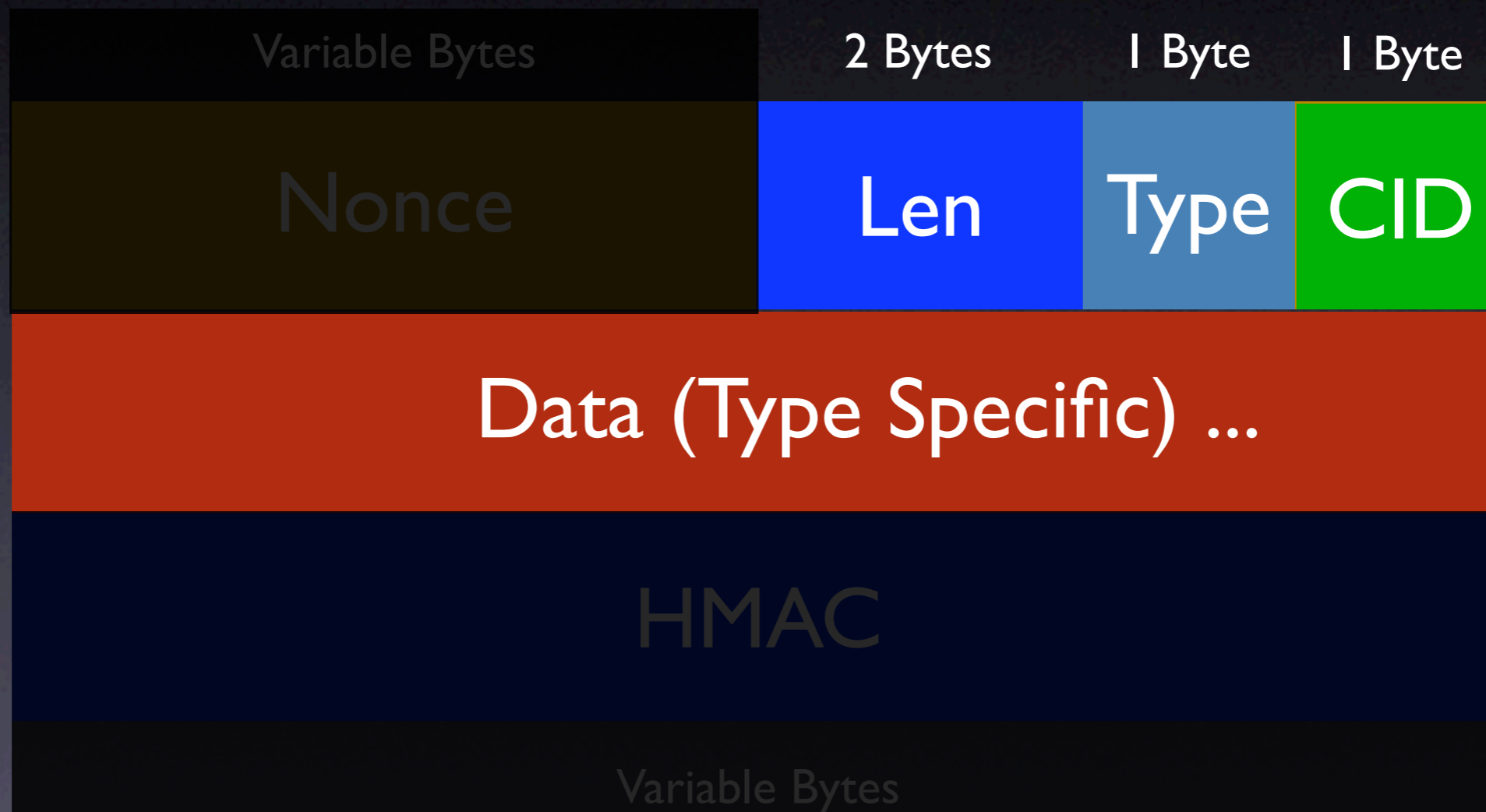
- Unrestricted by window size
- Easy to turn into Bittorrent-style requests
- Easy recovery from halted transfers
- Potentially good performance across high-latency networks (not yet tested, insert salt here)

CUTLASS

Cryptography

- Attackers
- Attacks
- Handshaking Protocol

CUTLASS Packet Encrypted Portions



Attacker Lineup

- Nosy ISPs
- Cops
- Competitors
- CIA, NSA, KGB, KFC, TLA

Attacks

- Key recovery
- Man in the middle
- Traffic analysis
- Direct cryptanalysis

Key Recovery

- Steal or own your machine
- Subpoena/search warrant
- RIP act

Key Recovery Countermeasures

- Ephemeral Diffie-Hellman key exchange
- Password-protected private keys

Man In The Middle

- Own the border router, legally or surreptitiously
 - Dsniff
 - CenterTrack
 - Phrack 56: “Things to do in Ciscoland when you’re dead”
- Attack does not scale to the whole Internet

MITM

Countermeasures

- SSH-style trust relationships
- Provides semi-easy “Out Of Band” key verification
- Not reinventing PKI

Traffic Analysis Countermeasures

- Traffic Tunneling/Onion Routing
- Chaffing
- Padding

Direct Cryptanalysis

- Key leakage
- Replay attacks
- Initialization vector collisions
- Cryptographic primitive weaknesses

Cryptanalysis

Countermeasures

- Default: Ephemeral AES-CTR via Diffie-Hellman, RSA, HMAC
- No weak algorithms
- 128 bit initialization vectors
- Sequence numbers
- Differing keys for each direction

Key Exchange

Initiator / "Client"

----- $\text{nonce}_c, H(\text{nonce}_c, \text{RSA}_s), \text{RSA}_c$ ----->

<----- $\text{nonce}_s, \text{nonce}_c, \text{RSA}_s$ -----

----- $\text{DH}_c, \text{SIG}_c(\text{DH}_c)$ ----->

<----- $\text{DH}_s, \text{SIG}_s(\text{DH}_s)$ -----

Responder / "Server"

The Five Year Plan

- DTLS - TLS over datagram (IETF draft)
- OpenPGP and SRP authentication for TLS (IETF drafts)
- DTLS + SRP + OpenPGP = sweet
- The IETF isn't quick, and I code even slower

CUTLASS Internals

Core Data Structures

- `cutlass_t` “Cutlass Handle”
 - Local keys, configuration information
 - Sockets
 - Hash tables of connections, directory entries
 - Muticies (Mutexen? Mutecii?)

CUTLASS Internals

Core Data Structures

- `conn_t` “Connection Handle”
 - Remote and session keys
 - Capabilities
 - Socket (if unique)
 - Audio and transport buffers and info
 - Detected MTU, etc.

CUTLASS Thread Model

Listener Thread

(Sometimes) Audio

(Sometimes) Audio

Housekeeping Thread

Action Handler

UI Thread

Action Handler

Connections passed via fingerprint

CUTLASS API

- CUTLASS is currently divided into libcutlass and clients
- Action handling functions registered by clients
- Existing clients: text-cutlass, gtk-cutlass

API Flow

- Call `cutlass_init()`
- Modify default values (if desired)
- Register action handlers
- Call `cutlass_start()`
- Go handle user input, action handlers will be called
- Call `cutlass_shutdown_all()`

Modifying Default Values

- Port: `cutlass_set_port()`
- Nickname: `cutlass_set_nick()`
- Permissions: `cutlass_set_permission()`
 - etc, etc
- Load private keys with `load_private_key()`, generate with `generate_rsa_key()`

Action Handlers

- Kicked out by internal threads asynchronously
- Event types:
 - User connected/disconnected
 - User offers a file
 - File transfer complete
 - And more!

Using Action Handlers

- Registering action handlers strictly optional
- Unhandled actions routed to /dev/null
- Each function provided the CUTLASS handle and a structure containing action information

Test Suite

- Test suite lives in /test subdirectory
- Checks handshaking, messages, file transfer
- No audio checks

Documentation

- `action_handler_guide.txt` - list of all actions and available information
- `api.txt` - API usage guide
- `goals.txt` - project goals
- `internals.txt` - thread locking policy and program structure
- `protocol.txt` - key exchange, cryptography, transport layer etc

The Future of CUTLASS

- Group management
- Windows, Mac OS X, and PocketPC clients
- Activate directories in clients
- Connection forwarding / Onion routing
- Video
- Gaim plugin

CUTLASS Groups

- Groups can be authenticated or unauthenticated
- Groups can be advertised or hidden
- Group communication is still point-to-point
- Group members are a consensus reality

CUTLASS Group Management

- SuperOps have copy of private group key
- SuperOps cannot be revoked
- Regular Ops can be designated by SuperOps, will not have private key
- Ops may authenticate users, kick, ban, etc.
- These are effectively suggested local policies

CUTLASS Directory Servers

- Anyone can be a directory server
- Store registered users, key fingerprints, and network locations
- Store advertised groups, group key fingerprints, and group operators
- Will NOT store file directories
- Will NOT be initially meshed, but is certainly a future desire

CUTLASS Connection Forwarding

- By default, clients will permit channel allocation between any two remote hosts
- Hosts may (and should) rate-limit between forwarded hosts
- Forwarded connections are opaque to forwarding host
- Peers may request peers to “meet” at a designated hosts

Defend The Bill of Rights!



Get CUTLASS Today!

<http://www.synacklabs.net/projects/cutlass/>

cutlass-subscribe@synacklabs.net

<http://cutlass.info>