Secret Handshake

x509 Based Malware
Command & Control Channel

John Conwell

@turbocodr

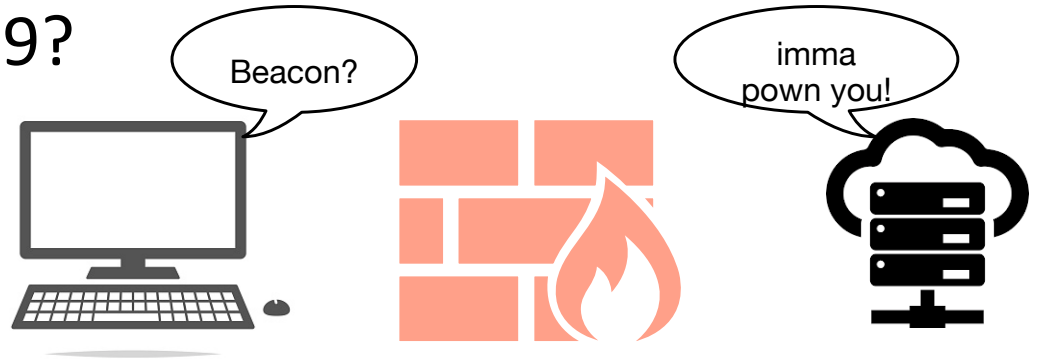
https://iamjohn.me

Background

- Data Science
- InfoSec / threat actor infrastructure hunting
- Soooo many x509s
- Never found anything groundbreaking
- Rumors of red team usage
- Holy Grail would be a malware C2 channel

Malware Command & Control (C2)

- Server sends commands to infected clients
- Client responds back to server
- Beaconing
- Download binaries from C2 server
- Exfiltrate data to C2 server
- Blend into background network traffic
- HTTP/S, DNS, UDP, many others…x509?

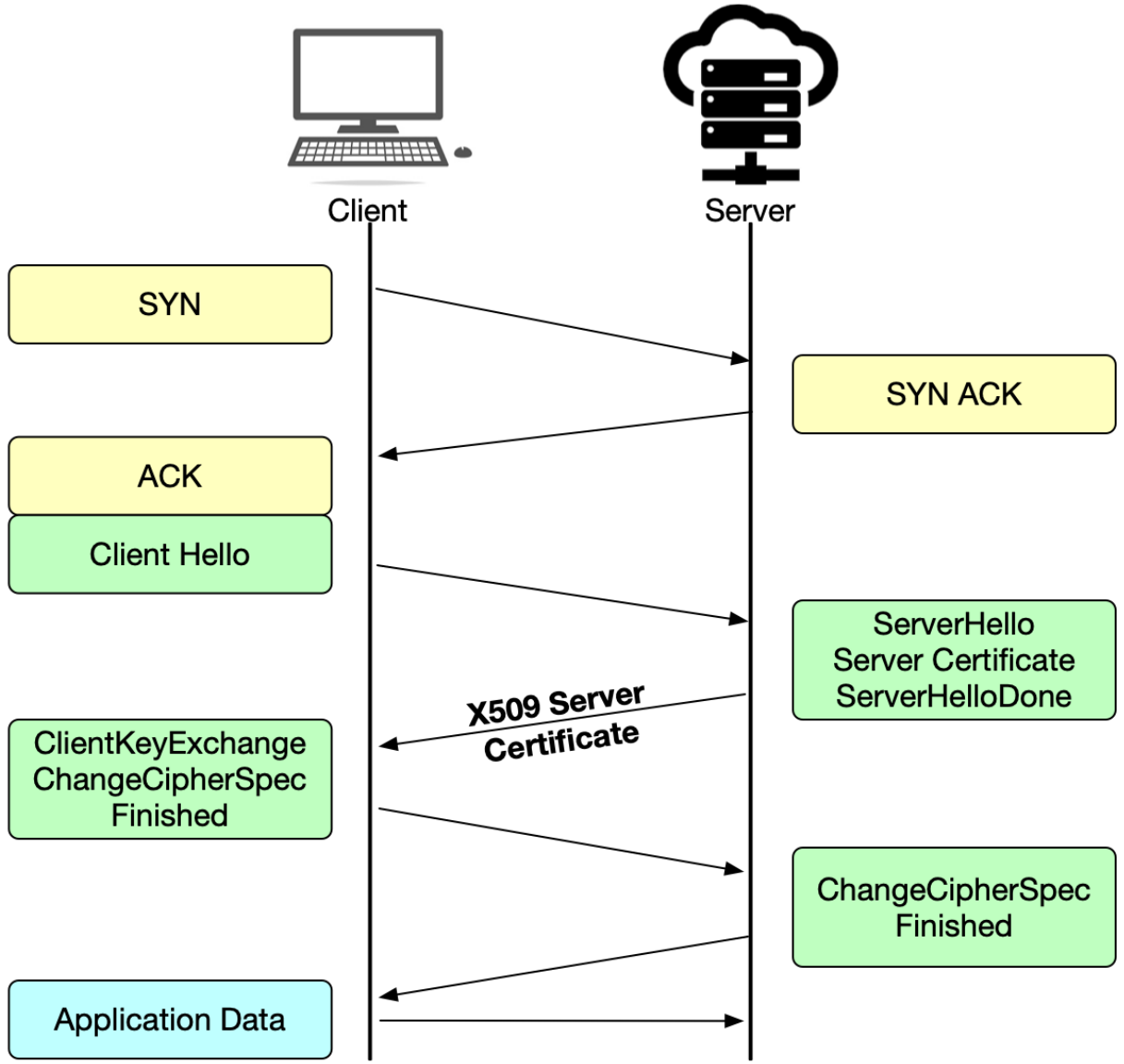


TLS Everywhere

- Google: 95% of internet traffic uses HTTPS
  - https://transparencyreport.google.com/https/overview?hl=en
- x509 certificates enable TLS/HTTPS
- x509 certs are just files
- North-South network logs are littered with TLS session
- NDRs don’t focus on x509 certs
- x509 certs are issued by Certificate Authority
  - Verisign, Let’s Encrypt
  - Issue your own root cert

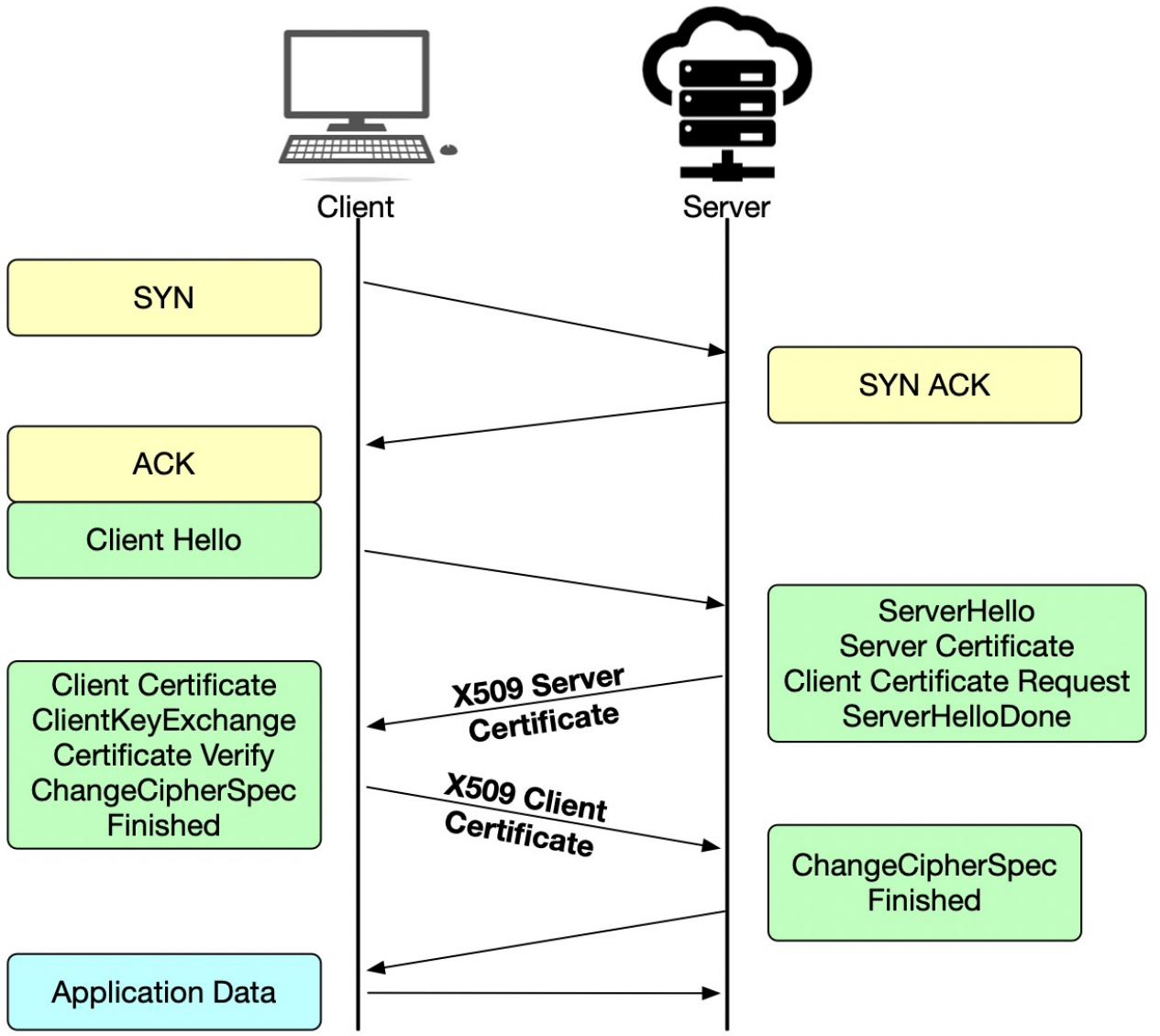
TLS Authentication

- Server sends x509 cert to client
- Client authenticates cert is issued by a trusted Certificate Authority
- Client has no way to send response back to server
- Won’t work for Request-Response Channel



Mutual TLS Authentication

- Both server & client exchange x509 certs
- Client authenticates server cert
- Server authenticates client cert
- Server & client certs can be issued by same CA cert



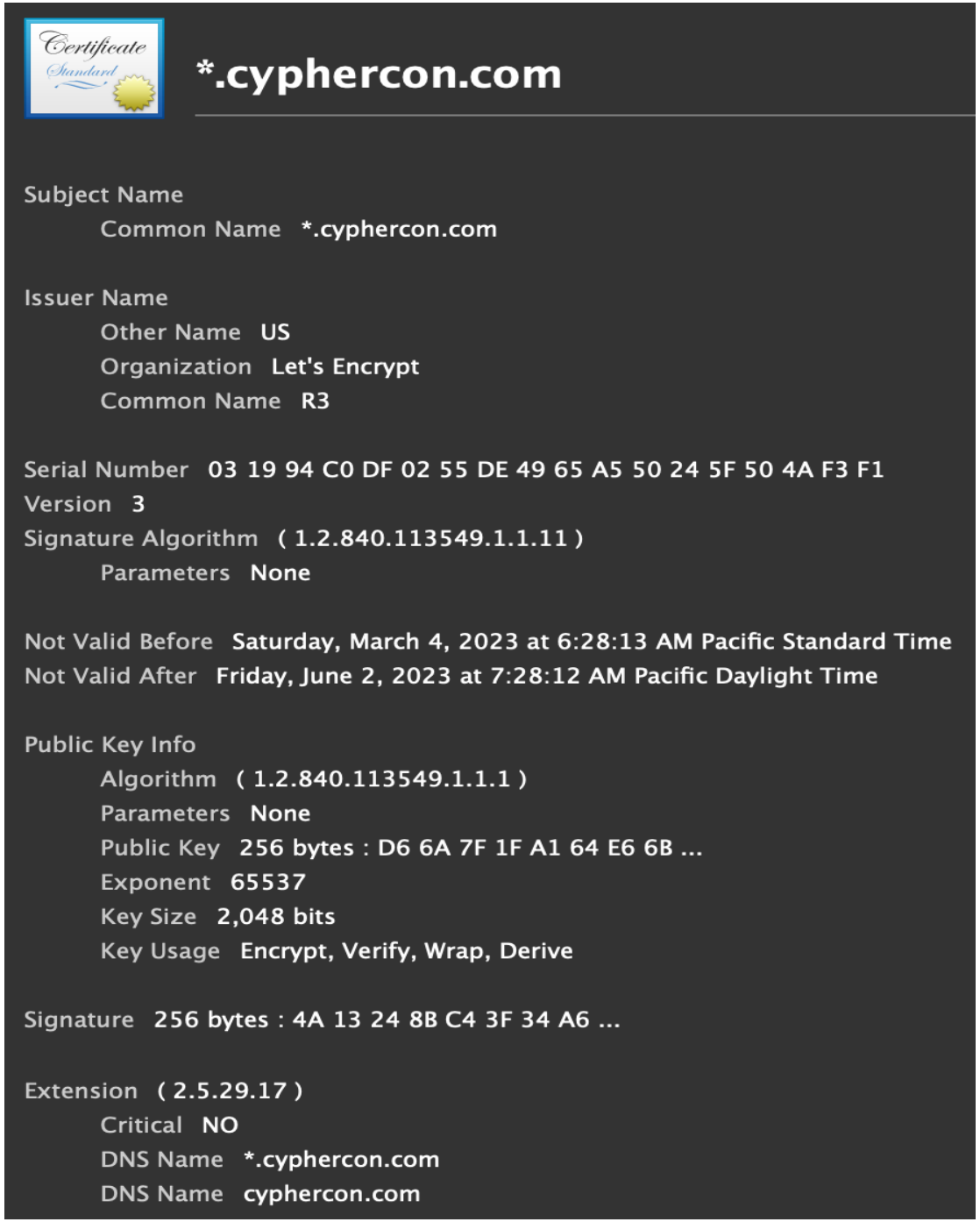
mTLS x509 Based C2 Channel

- Server and the client exchange x509 certs
- Opportunity to create a request-response communication channel
- Challenges
  - Embed data into x509 certificates
  - Design a request-response communication pattern
  - Dynamically issue new x509 certs for each mTLS handshake
  - Network sockets created each handshake

Embedding Data In x509 Certs

Hide data + valid TLS session

- Issuer (Distinguished Name)
- Algorithm Fields
- Serial Number
- Subject (Distinguished Name)
- Subject Alt Name
- Other x509 Extensions



Embedding Data In x509 Certs

x509 Extension Formatting

- ASN.1 encoding
- Specific encoding format
- Format associated with OID
- Must create valid TLS session
- *ASN.1 null field has a length property*

x509 Extensions

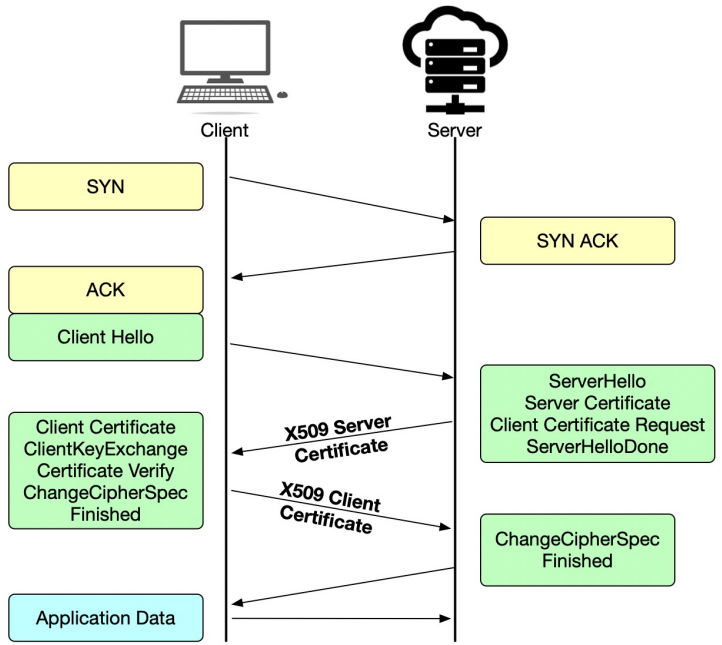
- subjectKeyIdentifier
- basicConstraints
- nsComment (deprecated)
- “Arbitrary Extension”
  - OID not registered

2 Fields Added To x509 Cert

- Command text field: nsComment x509 extension
  - Client: ‘Beacon’, ‘Response’
  - Server: ‘Sleep’, ‘Bash’, ‘Exfil’, ‘Kill’
- Data payload: Arbitrary Extension
  - Server: Bash command to run
  - Client: Bash command output
  - Server: Binary Dropper
  - Client: Data exfiltration
  - OID: 1.2.643.5.1.8.666

mTLS/x509 Communication Channel

- Request-Response Communication Pattern
- Server command -> Client command output
- Certs needed when sockets are created
- Requires 2 mTLS handshakes
- Pseudo-half duplex transmission mode



x509/TLS Communication Channel

Client

Server

Request Message	1) x509 Generation (beacon cert)		1) x509 Generation (command cert)
Response Message			

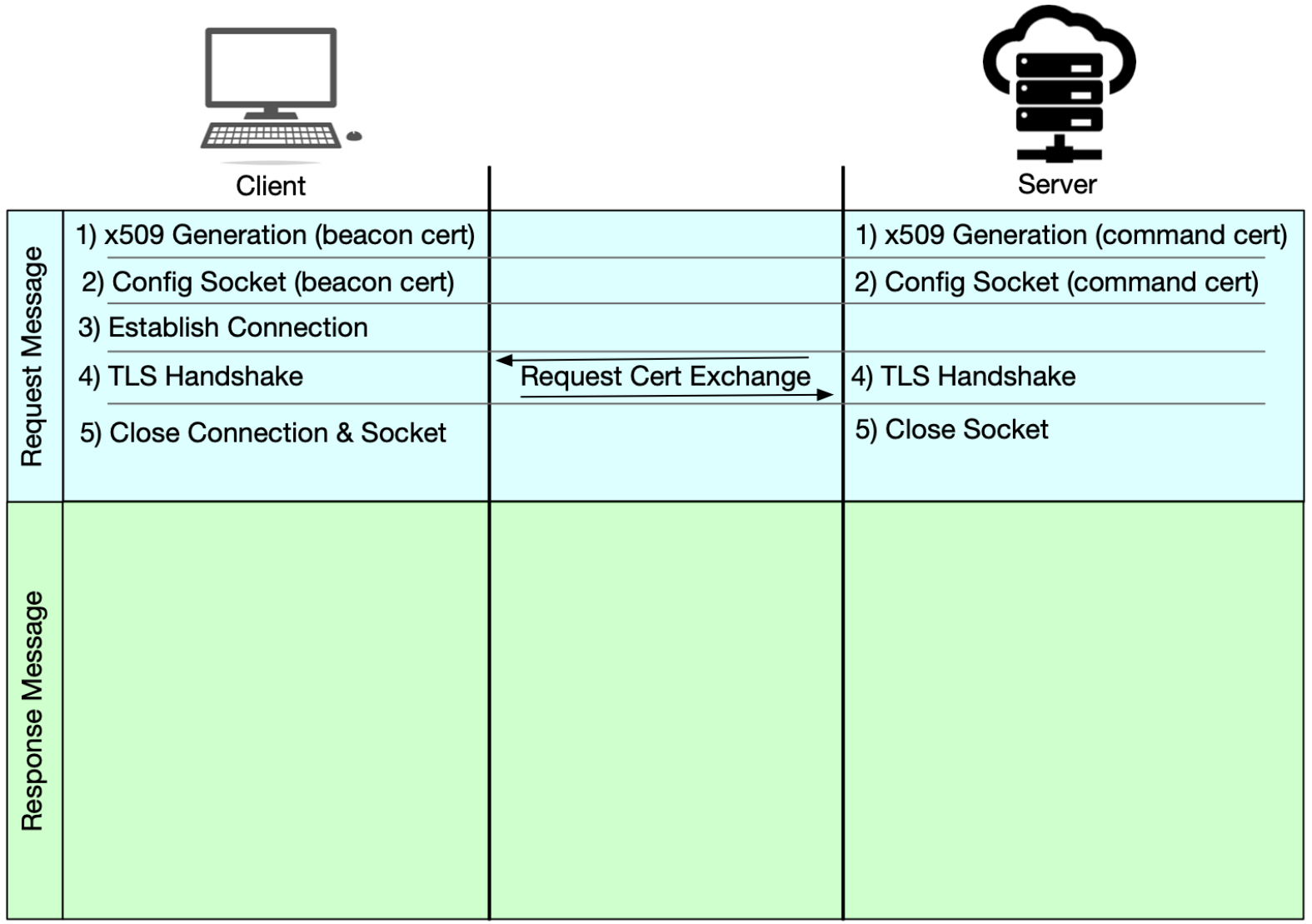
x509/TLS Communication Channel

Client

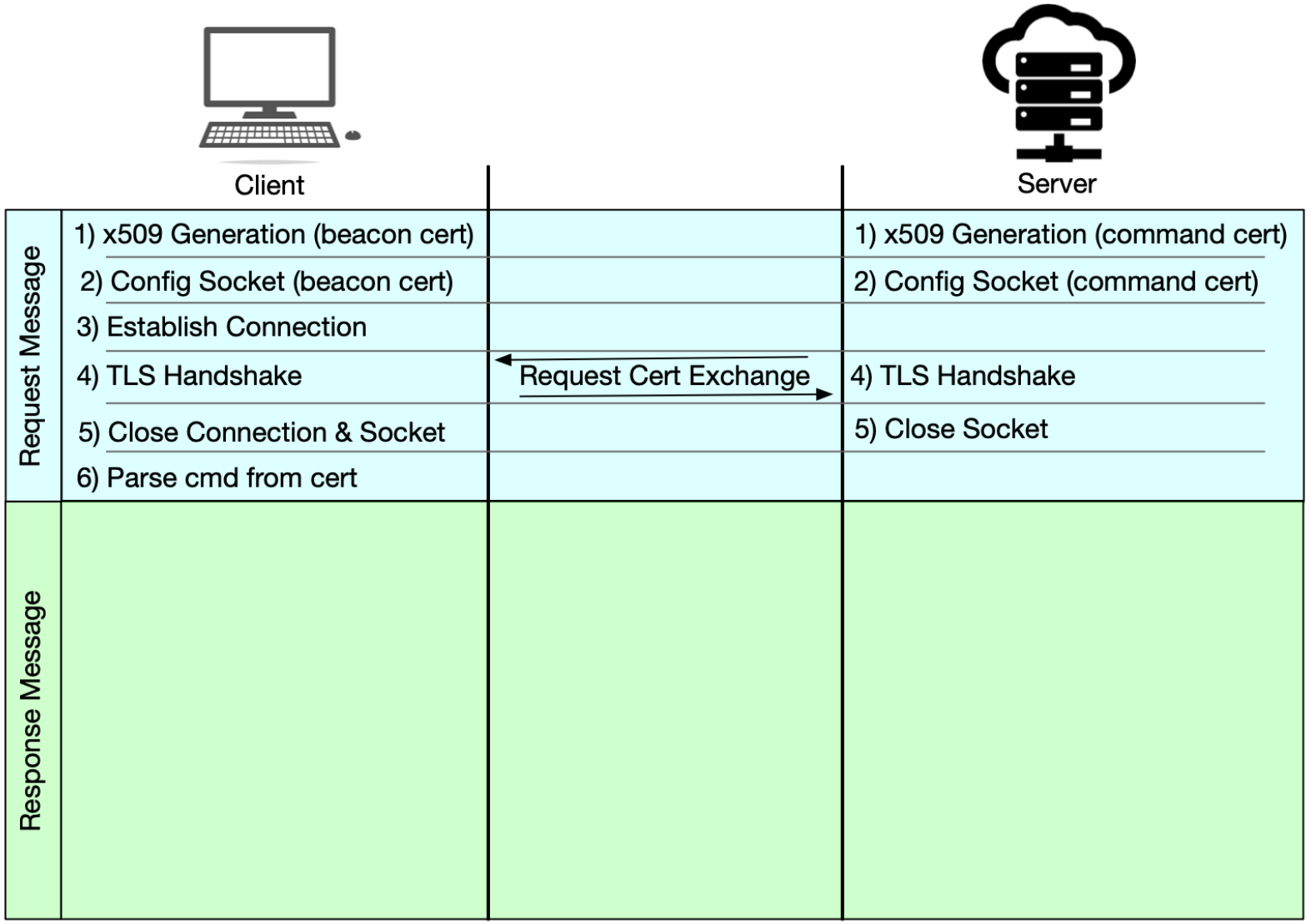
Server

	Client		Server
Request Message	1) x509 Generation (beacon cert) 2) Config Socket (beacon cert)		1) x509 Generation (command cert) 2) Config Socket (command cert)
Response Message			

x509/TLS Communication Channel



x509/TLS Communication Channel



x509/TLS Communication Channel

	Client		Server
Request Message	1) x509 Generation (beacon cert)		1) x509 Generation (command cert)
	2) Config Socket (beacon cert)		2) Config Socket (command cert)
	3) Establish Connection		
	4) TLS Handshake	Request Cert Exchange	4) TLS Handshake
	5) Close Connection & Socket		5) Close Socket
	6) Parse cmd from cert		
Response Message	7) Execute cmd, collects output		

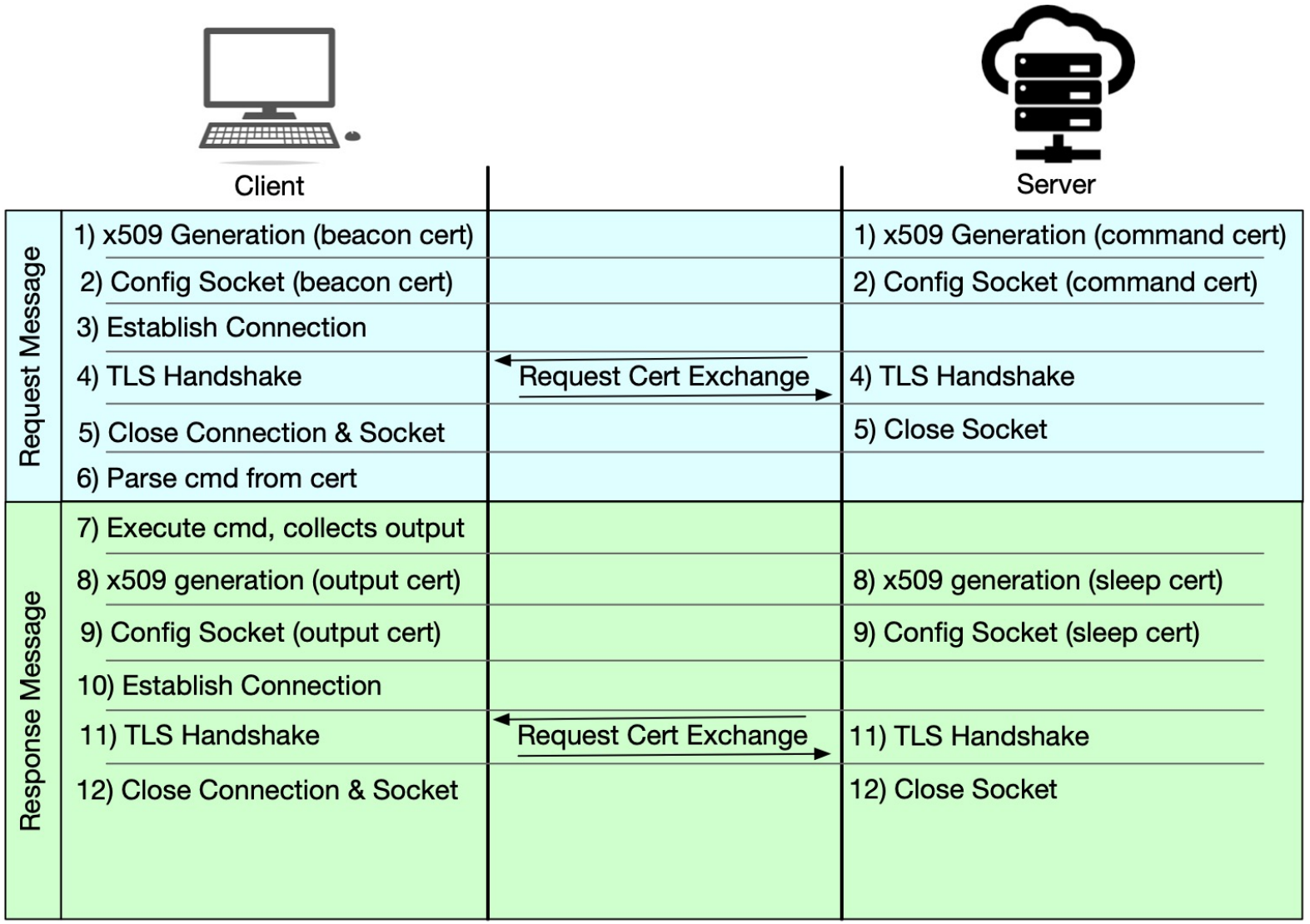
x509/TLS Communication Channel

	Client		Server
Request Message	1) x509 Generation (beacon cert)		1) x509 Generation (command cert)
	2) Config Socket (beacon cert)		2) Config Socket (command cert)
	3) Establish Connection		
	4) TLS Handshake	Request Cert Exchange	4) TLS Handshake
	5) Close Connection & Socket		5) Close Socket
	6) Parse cmd from cert		
Response Message	7) Execute cmd, collects output		
	8) x509 generation (output cert)		8) x509 generation (sleep cert)

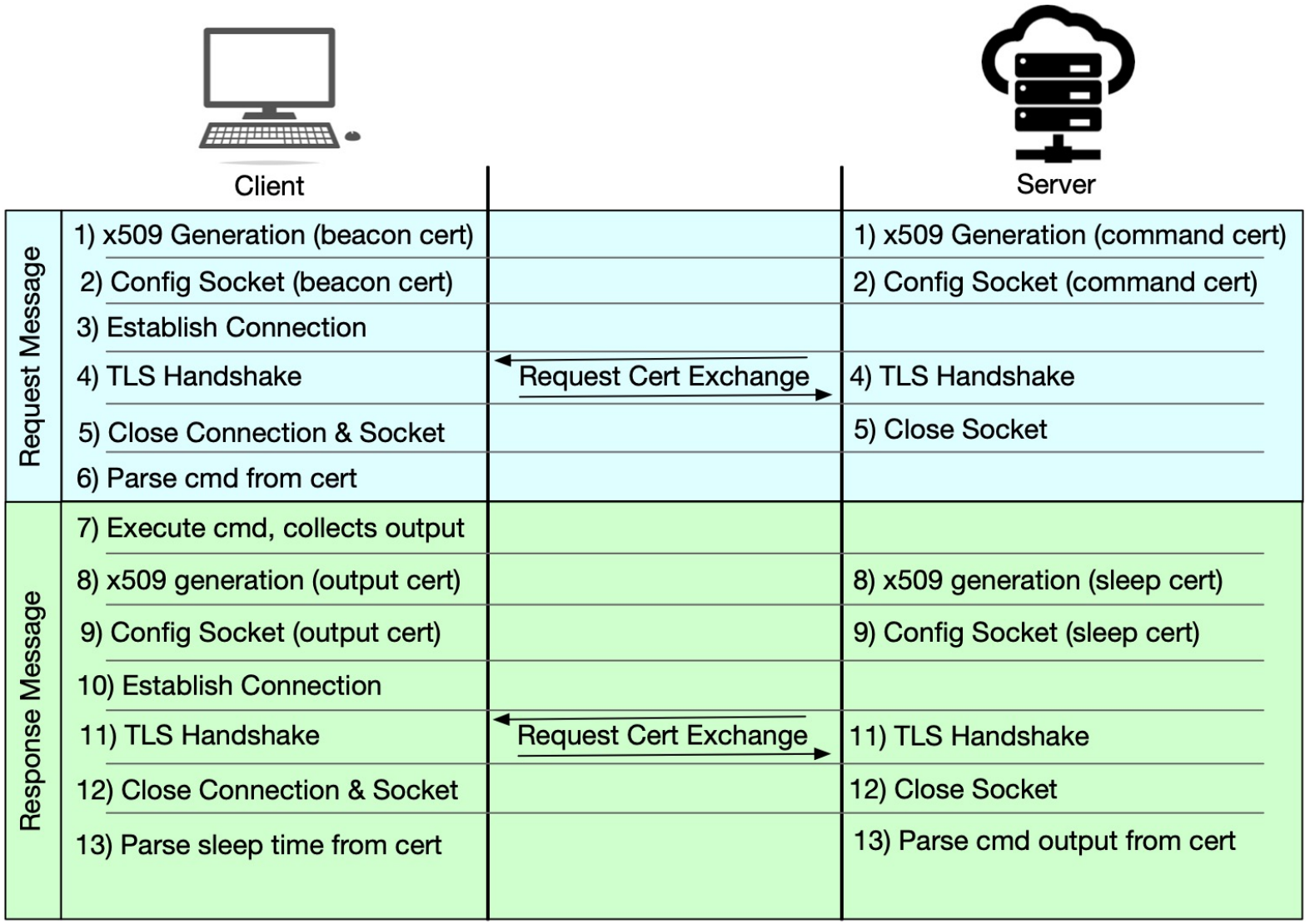
x509/TLS Communication Channel

	Client		Server
Request Message	1) x509 Generation (beacon cert)		1) x509 Generation (command cert)
	2) Config Socket (beacon cert)		2) Config Socket (command cert)
	3) Establish Connection		
	4) TLS Handshake	Request Cert Exchange	4) TLS Handshake
	5) Close Connection & Socket		5) Close Socket
	6) Parse cmd from cert		
Response Message	7) Execute cmd, collects output		
	8) x509 generation (output cert)		8) x509 generation (sleep cert)
	9) Config Socket (output cert)		9) Config Socket (sleep cert)

x509/TLS Communication Channel

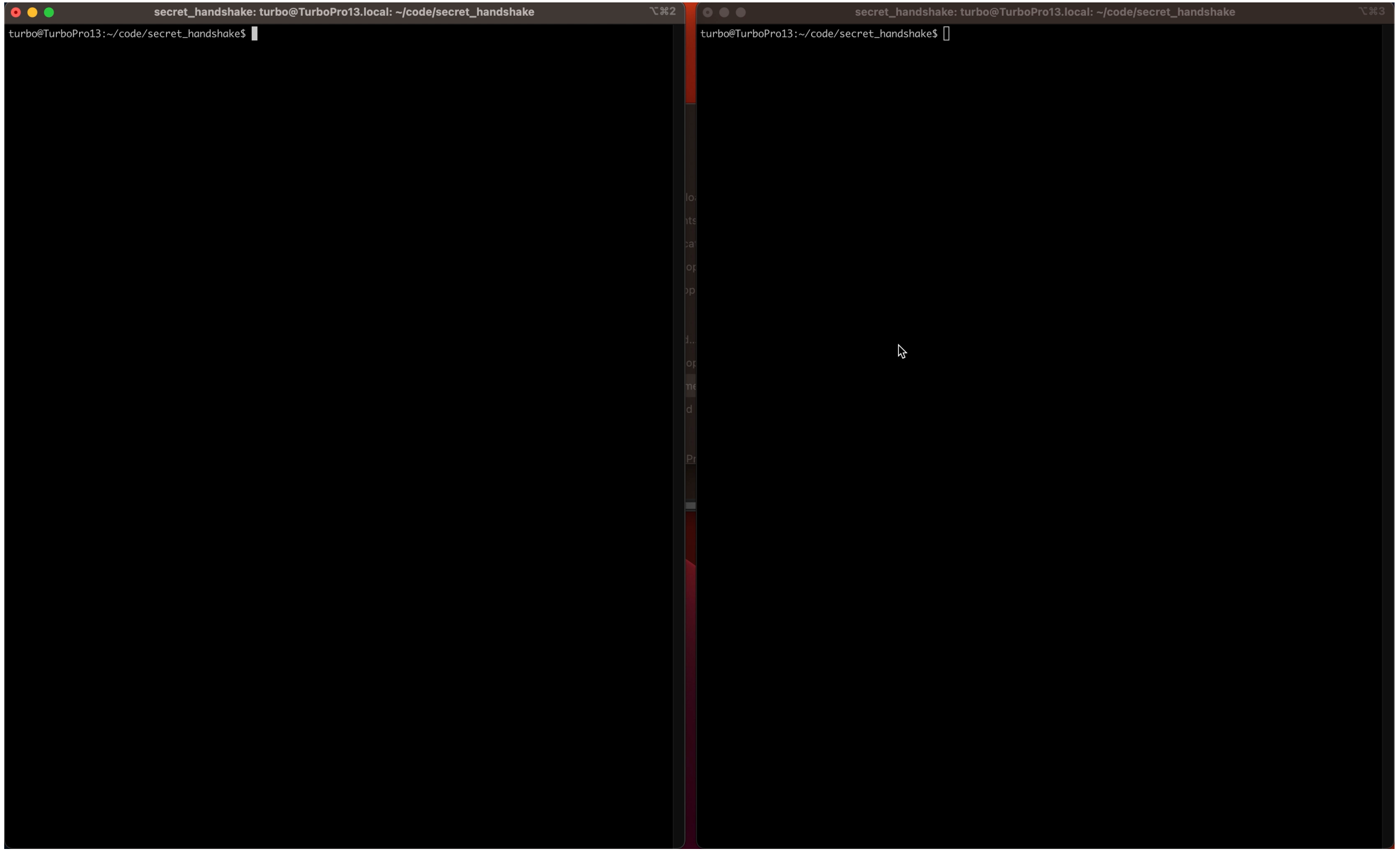


x509/TLS Communication Channel



x509/TLS Communication Channel

	Client		Server
Request Message	1) x509 Generation (beacon cert)		1) x509 Generation (command cert)
	2) Config Socket (beacon cert)		2) Config Socket (command cert)
	3) Establish Connection		
	4) TLS Handshake	Request Cert Exchange	4) TLS Handshake
	5) Close Connection & Socket		5) Close Socket
	6) Parse cmd from cert		
Response Message	7) Execute cmd, collects output		
	8) x509 generation (output cert)		8) x509 generation (sleep cert)
	9) Config Socket (output cert)		9) Config Socket (sleep cert)
	10) Establish Connection		
	11) TLS Handshake	Request Cert Exchange	11) TLS Handshake
	12) Close Connection & Socket		12) Close Socket
	13) Parse sleep time from cert		13) Parse cmd output from cert
	14) Sleep		


secret_handshake: turbo@TurboPro13.local: ~/code/secret_handshake
⌥⌘2
turbo@TurboPro13:~/code/secret_handshake$
secret_handshake: turbo@TurboPro13.local: ~/code/secret_handshake
⌥⌘3
turbo@TurboPro13:~/code/secret_handshake$

C2 Channel Detection

I never want to release something potentially malicious without providing detection logic

- Certs *not* issued by trusted Certificate Authorities
- Many mTLS sessions between same host/dest IP & port
- Many mTLS sessions in close time proximity
- Look for sleep+jitter periodicity between session
- Cert fingerprints will be different for each mTLS session
- Cert byte sizes will vary per session

C2 Channel Detection

I never want to release something potentially malicious without providing detection logic

- Certs *not* issued by trusted Certificate Authorities
- Many mTLS sessions between same host/dest IP & port
- Many mTLS sessions in close time proximity
- Look for sleep+jitter periodicity between session
- Cert fingerprints will be different for each mTLS session
- Cert byte sizes will vary per session
- Look for occasional repeated cert fingerprints

Should be easy to detect…

Detection…It’s Never That Easy

Set of enterprise services that have similar NS mTLS traffic pattern

- Tanium
- Microsoft System Center Configuration Manager (SCCM)
- Microsoft Monitoring Agent
- Azure Hybrid Runbook Worker
- Palo Alto Networks
- MuleSoft
- TrustedSource
- Cohesity Helios
- EMC's Global Security Organization
- Alert Logic

Benign Enterprise Services

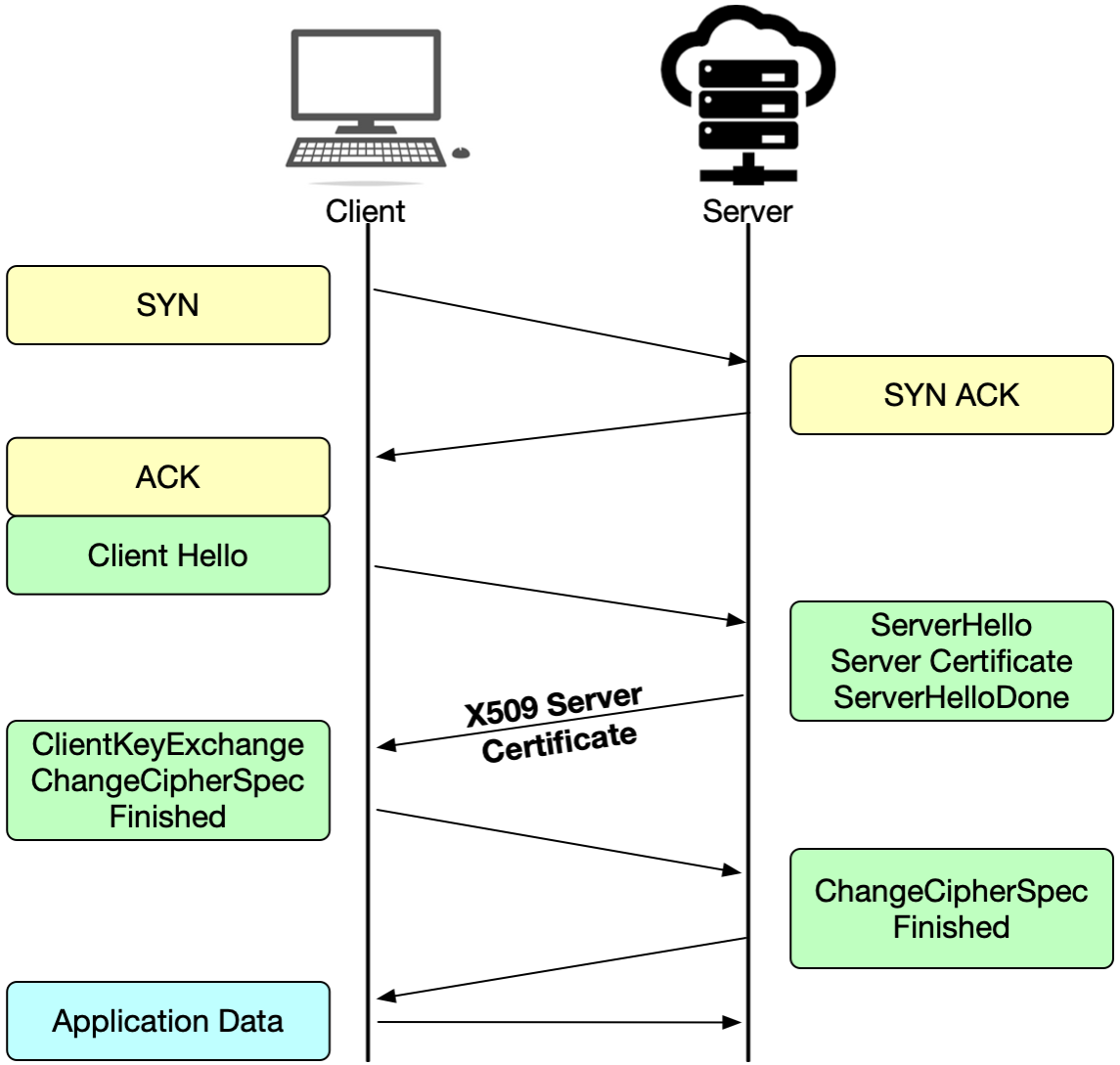
- Asset or device management services
- Issuer is company name
- Subject has asset id in key/value pair
- Sending same set of certs each time it runs
- Repeating all certs every *n* hours
- Server certs should be the same across all sessions

Mitigation

SSL Inspection

Potential TLS C2 Channel

- Random Bytes (32 bytes)
- Message ID (32 bytes)
- eSNI (65.79 kb)
- Custom TLS library on both client/host
- Super chatty, lots of TLS sessions
- SSL Inspection should block



Thank you!

Questions?

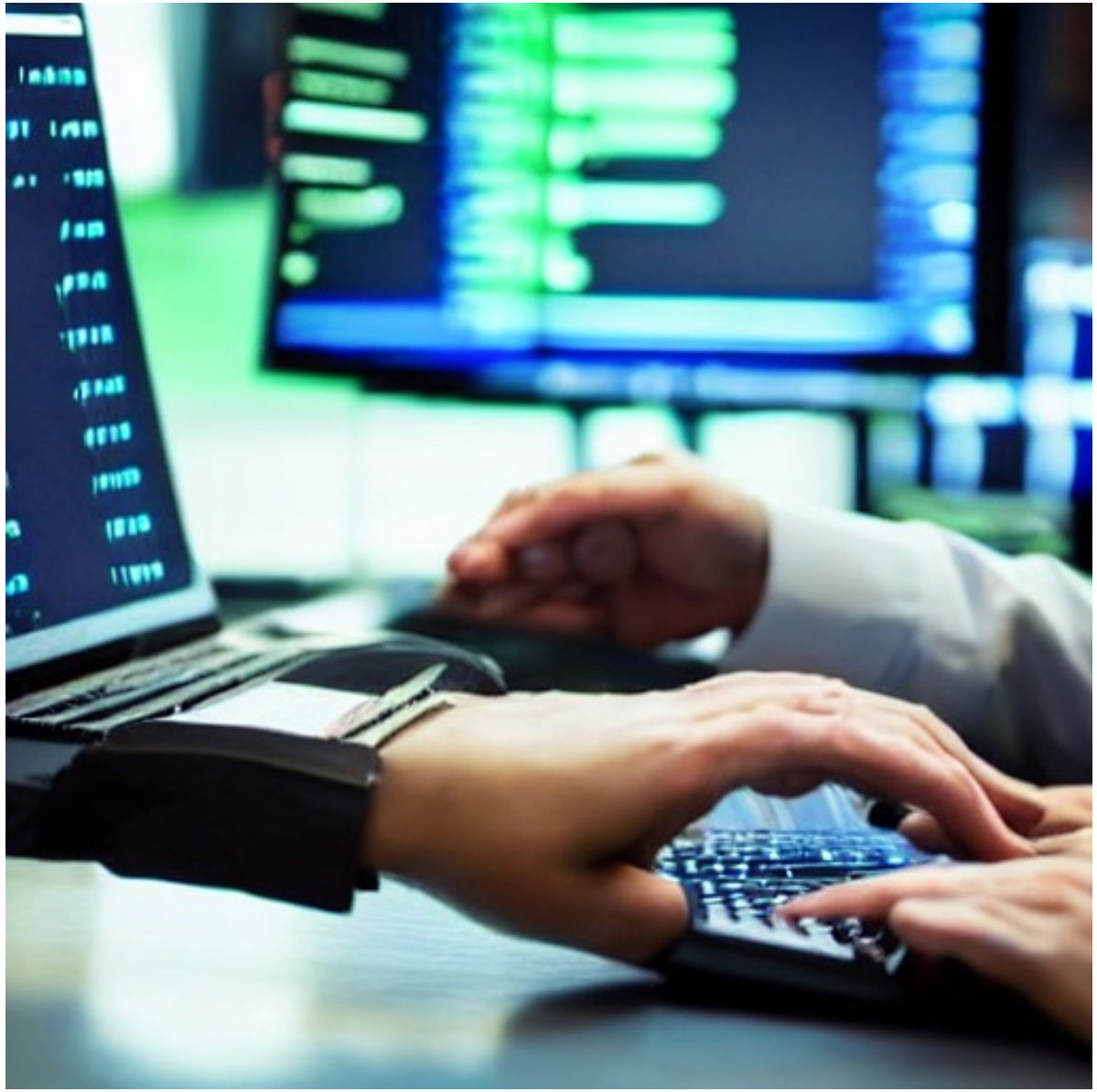
https://github.com/jconwell/secret_handshake

Prior Art Note: My idea was stolen before I ever had it (just kidding)
2018 Jason Reaves prototyped a TLS based Malware binary dropper in Go
2019 he wrote a white paper on this topic: https://vixra.org/pdf/1801.0016v1.pdf

Artwork generated by Stable Diffusion for “Secret Handshake”





Data Exfiltration

- Unauthorized transfer of data from one host to another
- Max payload size of ~64 kb
- Partition files into multiple x509 certs
- Repeat steps 8-13 above per partition
- Final payload cert has exfil termination flag

Damn This Is Slow!

- Data exfil generates a LOT of x509 certificates
- RSA private key size impacts cert generation speed
- 4096 bit key: slow-as-shit mb/sec
- 2048 bit key: 0.4 mb/sec
- 1024 bit key: 1.5 mb/sec