Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

A distributed deployment model for Encrypted Client Hello

Ted Johnson

Supervisor: Dr Stephen Farrell

April 2024

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Master in Computer Science (MCS)

Declaration

I hereby declare that this dissertation is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at http://www.tcd.ie/calendar.

I have completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write.

I consent to the examiner retaining a copy of the thesis beyond the examining period, should they so wish (EU GDPR May 2018).

Signed: ______________________ Date: ______________________

A distributed deployment model for Encrypted Client Hello

Ted Johnson, Master in Computer Science

University of Dublin, Trinity College, 2024

Supervisor: Dr Stephen Farrell

Encrypted Client Hello (ECH) is a proposed extension to the Transport Layer Security (TLS) protocol that encrypts information currently leaked during connection, which is now beginning to see implementation and adoption on the Internet. However, typical deployment strategies encourage placing many TLS servers behind a single ECH-service provider to form an anonymity set, which introduces significant network centralisation and limits the types of environments the extension can be operated in.

This report presents a method for distributing the deployment of ECH amongst a loose network of co-operating TLS servers, such that each server functions as an ECH-service provider for all others. This allows for ECH-enabled clients to access services through any co-operating TLS server, greatly strengthening service availability and network flexibility. The effectiveness of this solution is evaluated based on its privacy and security implications, impact to overall network performance and ability to fairly allocate load between participating servers over time.

Acknowledgements

This work has been completed under the guidance and persistence of Dr. Stephen Farrell, without which I cannot say there would be much of a paper to speak of. His continued efforts on the implementation of Encrypted Client Hello within OpenSSL and related work as part of the DEfO project has been fundamental to my research. Thank you.

Ted Johnson

University of Dublin, Trinity College
April 2024

Contents

List of Figures

List of Listings

1 Introduction

Encrypted Client Hello (ECH) is a proposed extension to the Transport Layer Security protocol version 1.3 (TLS 1.3) which has begun to see implementation and adoption on the Internet [1–3]. ECH seeks to allow encryption of the ClientHello message, which can contain potentially sensitive information such as the Server Name Indication (SNI) and Application-Layer Protocol Negotiation (ALPN) extensions. This is partially achieved through serving many private domains behind a common provider to form an anonymity set that conceals the true domain requested by the client.

Due to this, ECH introduces significant centralisation to the Internet. This paper presents a practical model for the distributed deployment of ECH amongst several co-operating TLS servers, where each server operates both as the origin server of its own domains as well as an ECH provider for other participating servers. The model addresses a number of implementation challenges, predominately related to ensuring the security of the protocol is not compromised and minimising the performance impact to the connection while strengthening service availability.

Included in this paper is a review of the background technology and concepts relevant to the discussion of the deployment model. This is followed by a study of the model's design and the complications which influenced it. We then see how this design can be implemented within a practical scenario and discuss some of its deployment considerations. In the subsequent chapter, an analysis and criticism of both the results taken from this implementation and the design as a whole is used to assess the quality of the solution. Finally, I conclude the report with a summary of the work completed and delineate where future contributions could best benefit the further development of the deployment model.

1.1 Motivation

Nottingham has previously cautioned against the introduction of centralisation through Internet standards [4]. Of particular relevance to ECH is his highlight of the adverse effect centralisation can have on infrastructure resilience and service availability through reliance on a single entity. This is especially detrimental to ECH where the effectiveness of its anonymity set grows with the number of private domains served by a single provider. Nottingham also writes on susceptibility of centralisation to stifle "permissionless" innovation and induce an unhealthy monoculture, which may result in less overall technological progress and robustness of the ECH protocol.

Additionally, allowing entirely independent servers to co-operate from across the Internet to provide ECH support for each other enables several distinct organisations to work together to offer improved privacy for their users without the requirement for co-located servers nor the dependence of any on the availability on another. Consider here global networks of whistleblower services, investigative journalists and human rights non-profit organisations who share an interest in protecting the confidentiality of their members and users from persecution and retaliation.

For these reasons, the development of a model for the distributed deployment of ECH across several co-operating providers is a key step towards its broad adoption throughout the Internet and its application within more elaborate scenarios.

1.2 Project Objectives

The objectives of this research project can be summarised with the following question: "How can Encrypted Client Hello be deployed fairly amongst co-operating Transport Layer Security servers to reduce network centralisation without compromising the security of the protocol?" This task is composed of the following objectives:

1. **Identify principal challenges and appropriate solutions.** Before development can begin proper, we must first understand the environment the system would operate in and explore the technical and logistical issues it might face to determine the dominant criteria for design. We accomplish this through research and experimentation of the functioning of the protocol and its surrounding technologies.

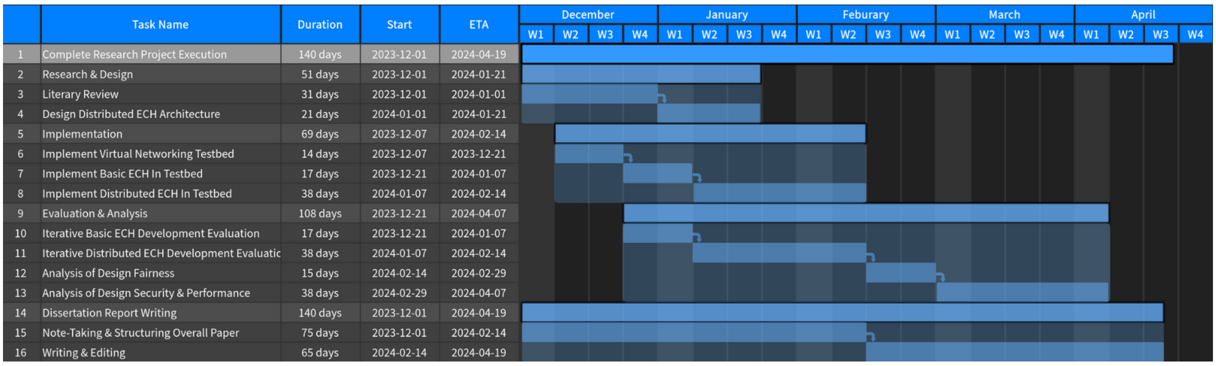
2. **Design, evalute and contrast deployment models.** An iterative development process is used to produce a series of incrementally improved skeletal prototypes, with the goal to rapidly design and test for functionality guided by the design criteria and results of previous work as heuristics.

3. **Analyse model implementatons through simulation.** Promising design solutions are fleshed out into full implementations within deterministic, reproducible and quantifiable simulated environments, where security and performance implications can be easily isolated and compared. This allows for these effects to be consistently measured against implementations based on a centralised ECH deployment model or with ECH support disabled entirely. It is also expected that unforeseeable practical

challenges and considerations are to be unveiled during this work.

4. **Conclude findings and present results.** The data collected and learnings gained during analysis of model implementations is to be compiled into a report on the overall effectiveness of distributed ECH deployment and recommendations for future researchers and service operators. Of particular use here is a study on the effect distributed ECH has on performance when compared to other implementations.

In preparation of these objectives, I produced the Gantt chart included in Fig. 1.2.1 to help gauge my progress during the four months of work. While in the final result I have found more emphasis has been placed on implementation, the overall structure of the timeline has been followed reasonable well.



	Task Name	Duration	Start	ETA
1	Complete Research Project Execution	140 days	2023-12-01	2024-04-19
2	Research & Design	51 days	2023-12-01	2024-01-21
3	Literary Review	31 days	2023-12-01	2024-01-01
4	Design Distributed ECH Architecture	21 days	2024-01-01	2024-01-21
5	Implementation	69 days	2023-12-07	2024-02-14
6	Implement Virtual Networking Testbed	14 days	2023-12-07	2023-12-21
7	Implement Basic ECH In Testbed	17 days	2023-12-21	2024-01-07
8	Implement Distributed ECH In Testbed	38 days	2024-01-07	2024-02-14
9	Evaluation & Analysis	108 days	2023-12-21	2024-04-07
10	Iterative Basic ECH Development Evaluation	17 days	2023-12-21	2024-01-07
11	Iterative Distributed ECH Development Evaluatic	38 days	2024-01-07	2024-02-14
12	Analysis of Design Fairness	15 days	2024-02-14	2024-02-29
13	Analysis of Design Security & Performance	38 days	2024-02-29	2024-04-07
14	Dissertation Report Writing	140 days	2023-12-01	2024-04-19
15	Note-Taking & Structuring Overall Paper	75 days	2023-12-01	2024-02-14
16	Writing & Editing	65 days	2024-02-14	2024-04-19

Figure 1.2.1: Predicted timeline of project as of the 7th of December, 2023.

1.3 Research Contributions

This work provides evidence for the viability of the distributed deployment of ECH between co-operative TLS servers. It supports its argument that the deployment model presented does not compromise the security of the protocol and minimises impact to network performance though analysis of the data produced by implementations within simulated networking environments. Additionally, an evaluation of several traffic masking and normalisation techniques is given to serve as the bases for further work on disrupting traffic correlation attacks applicable to ECH and elsewhere. Finally, the delivered project may also contribute academic value as a deterministic, reproducible tutorial on the deployment and operation of ECH using commonplace software and tooling.

2 Background

This chapter offers an overview of the technology and concepts needed to understand the context and relevance of the work within the broader world. The review is conducted predominately through a networking, security and privacy perspective to best highlight the aspects pertinent to the distributed deployment of ECH. This chapter also represents the bulk of the effort put into investigating and studying the functioning of ECH while identifying and experimenting with different deployment models.

The contents of this chapter include a high level description of the Transport Layer Security protocol and the Domain Name System, with a more detailed look at the components that enable ECH functionality. This is followed by an inspection of ECH itself, its security properties and the mechanisms which allow for distributed deployment. Finally, we survey how a variety of traffic analysis techniques that can be used to infer sensitive information from patterns in network activity, as well as the countermeasures which exist to mask these patterns.

2.1 Transport Layer Security

Transport Layer Security (TLS) is a cryptographic protocol proposed by the Internet Engineering Task Force (IETF) which enables secure communication over public networks. Applications and services can establish an encrypted communication channel to transmit private information such that confidentiality, integrity and authenticity of the data can be ensured. TLS is commonly used to protect Internet traffic, having seen widespread adoption and several revisions since its original inception in 1999, superseding the Secure Sockets Layer (SSL) specifications previously defined by Netscape Communications [5–7].

TLS is designed to operate on top of a reliable transmission protocol between a client and server, typically the Transmission Control Protocol (TCP) when used over the Internet. In order to prevent eavesdropping, tampering and message forgery, TLS includes a number of security features based on a number of cryptographic mechanisms:

Confidentiality: All service and application data exchanged between the client and server is encrypted as to make it indecipherable to any intermediate party which might be intercepting their communication. For example, consider the importance of protecting passwords, banking information and patient health records.

Data integrity: In a similar manner, cryptographic properties are used to guarantee transferred data cannot be modified during transmission. This is critical for safeguarding against input manipulation in consequential situations, such as while specifying fields for a financial transaction.

Authentication: TLS provides the ability for both peers to verify the identity of the other, ensuring privileged communication is only performed with the intended recipient. Such a condition is fundamental for establishing trust and confidence in any sensitive environment.

TLS 1.3 is the latest defined standard for the protocol, having been published in August 2018 and contributing to the deprecation of TLS 1.0 and TLS 1.1 in March 2021 [8, 9]. Lee, Kim, and Kwon have measured a comparatively rapid adoption rate, reporting support by 48% of Alexa top 1M sites by 2021, which is attributed largely to the growth of cloud hosting providers such as Cloudflare [10, 11]. The version introduces many major changes over TLS 1.2, including the addition of a zero round trip time resumption (0-RTT) mode, further encryption and optimisation of the handshake and removal of outdated cryptographic algorithms and security mechanism with all key exchanges now providing forward secrecy. A change of particular relevance to ECH is the encryption of the digital certificate received by the client to authenticate the server.

2.1.1 Digital Certificates

TLS uses X.509 digital certificates to make assertions on the identity of entities within the network using a chain of trust model, and are intrinsic to the authentication within the public key infrastructure used to initiate a secure TLS key exchange [12]. Without this assertion in place, a malicious party may insert itself into the middle of any TLS connection to perform a man-in-the-middle attack by replacing any public key with their own outside the knowledge of either peers. This invalidates the security of the key exchange and thus compromises any security offered by TLS. Therefore, to have any confidence in a secure connection, we must be able to trust the authenticity of received public keys by associating them with a trustworthy digital certificate.

It is not feasible to have a trusted party for every entity install a certificate for every other entity, as this becomes impractical within larger networks, in which certificates may be created and replaced. Instead, this trustworthiness is established through the associativity, where cryptographic signatures provide a mechanism for one certificate to attest to the validity of another as depicted in Fig. 2.1.1. A Certificate Authority (CA) may issue new certificates using their private key to produce a signature that can be authenticated using the public key present in their own certificate. Furthermore, the certificate of the CA was issued by its parent CA and contains a signature, which itself can be authenticated in a similar manner. In this way, certificates are organised into a hierarchical chain of trust, where the trustworthiness of a certificate is asserted by the trustworthiness of its issuer. This chain of trust continues until a root certificate issued by a root CA using a self-signed signature is encountered at the base of the hierarchy, which is implicitly trusted by all entities.

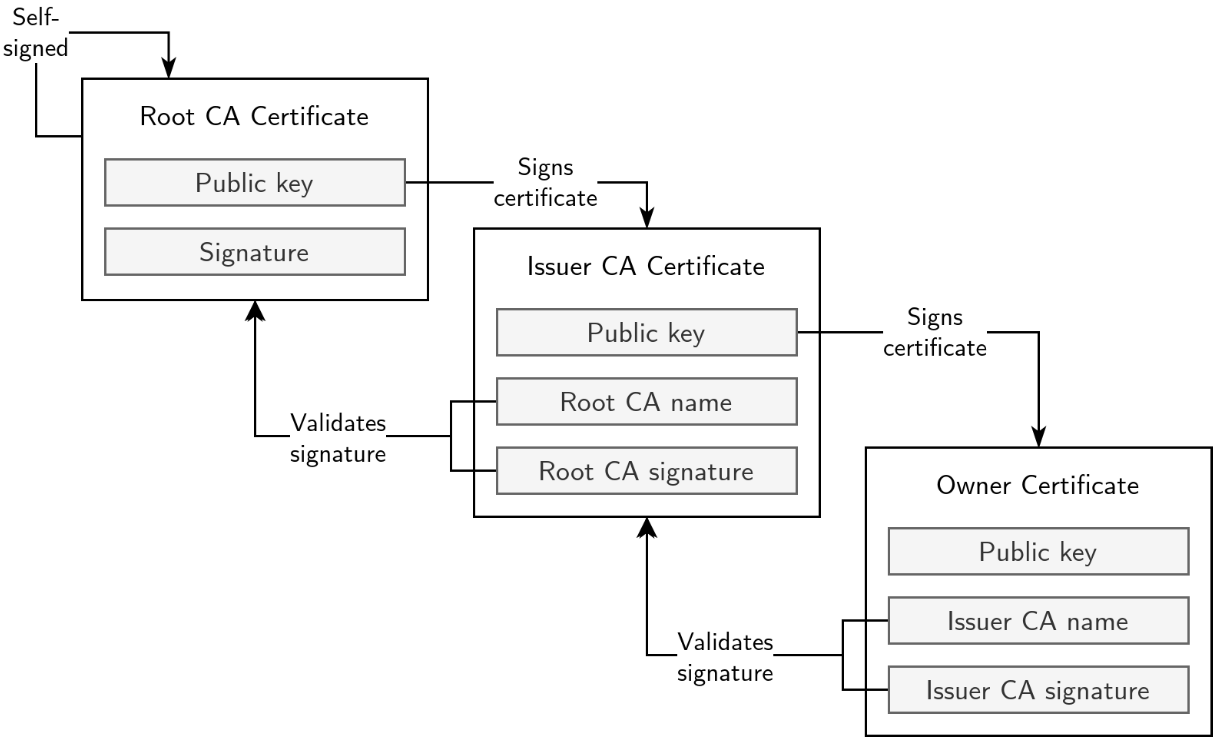


Figure 2.1.1: A chain of trust established between the unknown owner certificate and the implicitly trusted root CA certificate in order to authenticate the identity of the owner against its associated public key.

An organisation or individual must request new certificates from a CA using a Certificate Signing Request (CSR). The CA is then responsible for verifying the identity of the organisation or individual before issuing the certificate. To ensure the validity of certificates are consistent over time, X.509 certificates expire after a set period and must be renewed. Today, this renewal procedure has been widely automated using the Automatic Certificate Management Environment (ACME) protocol, which allows for web servers to complete challenges set by the CA to prove ownership of their domain name and public key without human involvement [13]. This has enabled a much shorter certificate rotation period, and it is common to see certificates set to expire within three months.

Through this process, an entity is only required to trust a few well-established root certificates to be capable of validating the authenticity of many certificates and their public keys. These root certificates are generally installed by an inherently trusted party, such as the device manufacturer or operating system, but new certificates may be installed by the user.

Typically on the Internet, it is only necessary for the identity of the server to be authenticated by the client, while the client remains unauthenticated to the server in the TLS context. In either case, certificates may be exchanged between the server and client during the TLS handshake.

2.1.2 TLS 1.3 Handshake

The TLS handshake is the series of messages exchanged between the client and server to establish the connection. It specifies the steps required to negotiate connection parameters, authenticate peer identities and yield a shared secret. TLS 1.3 was designed to improve the security and performance of the handshake over TLS 1.2 while reducing its overall complexity. Highlighted in these changes is the integration of parameter negotiation into the first client message, enabling encryption of much more of the handshake, as well as allowing application data to be sent by the client after only one round trip.

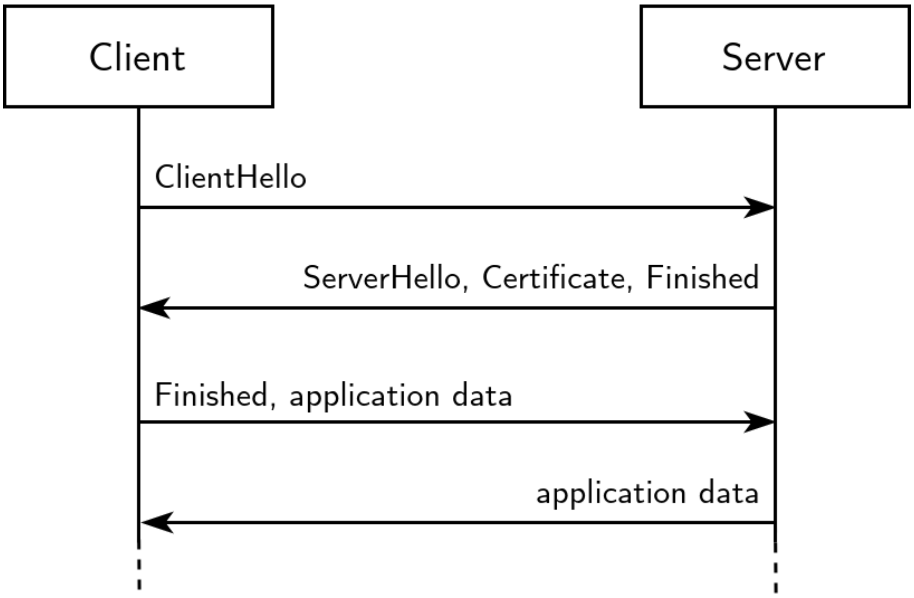

Client
Server
ClientHello
ServerHello, Certificate, Finished
Finished, application data
application data

Figure 2.1.2: Sequence diagram between a client and server describing a basic TLS 1.3 handshake with only server authentication.

Once a reliable transmission channel has been created between the client and server, a TLS 1.3 handshake can be performed as seen in Fig. 2.1.2. The core functionality of the handshake can be achieved in as few as four message types:

ClientHello: The client initiates the handshake by sending a ClientHello message without any encryption, containing information such as the supported TLS version number, along with a list of available cipher suites and their parameters for the server to choose from. Included in this is also an optimistic key share using the client's preferred cipher suite key exchange method, namely Diffie-Hellman (DH) or Elliptic Curve Diffie-Hellman (ECDH). Both of these generate ephemeral keys for each session, ensuring forward secrecy is preserved in the event the server's private key is compromised. Finally, the ClientHello message also contains a random value generated by the client used to prevent replay attacks.

ServerHello: If the server supports the client's preferred cipher suite, it is able to continue the key exchange immediately in the ServerHello message. Otherwise, the server must send a HelloRetryRequest to restart the key share with a different key exchange method which requires an additional round trip. The ServerHello message is sent without encryption and informs the client of what cipher suite and parameters were selected, as well as includes its own random value generated by the server. As the server has now completed its side of the key exchange, all subsequent communication is now encrypted using the selected symmetric encryption algorithm, such as AES-GCM or ChaCha20-Poly1305.

Certificate: The server then sends the client its certificate and proof of private key possession by signing a cryptographic hash of the transcript of the handshake so far. It may also choose to request authentication from the client using a CertificateRequest message, which requires the client to respond with its own Certificate message and proof of private key possession.

Finished: Finally, the server concludes its side of the handshake by initiating an exchange of Finished messages with the client. This message consists of a Message Authentication Code (MAC) over the cryptographic hash of the transcript of the entire handshake. In this way, the client can confirm success of the key exchange and integrity of the transaction. Once the client has received the ServerHello with the completed key exchange as well as decrypted and validated the Certificate and Finished message, it produces its own Finished message for the server to perform the same checks. Finally, with both peers in agreement on the security of the connection, application data can begin to be securely exchanged.

There are many more complexities to this handshake which are not particularly relevant here that have been omitted from this overview for the sake of brevity. However, one important topic to mention is the inclusion of extensions.

2.1.3 Extensions

Within both the TLS 1.2 and TLS 1.3 handshakes, the ClientHello and ServerHello messages may be extended with additional functionality, which allows the protocol to fulfil a wider range of use cases and accommodate evolving requirements. The usage of extensions has been significantly expanded in TLS 1.3 and now includes the ability for previously unencrypted ServerHello extensions to be placed within the new EncryptedExtensions message sent after ServerHello. Furthermore, a number of new extensions have been defined with several extensions now being mandatory to include in the TLS 1.3 handshake. Indeed, the Key Share extension is the provided mechanism for performing key exchanges and the Supported Versions extension is used to signify which versions of TLS is supported.

Nevertheless, the ClientHello message is not encrypted and all of its extensions are sent in the clear. Some of the ClientHello extensions include potentially sensitive information, such as the Server Name Indication (SNI) and Application Layer Protocol Negotiation (ALPN) list. It is this privacy weakness that the purposed ECH extension is attempting to remedy.

2.2 The Domain Name System

The Domain Name System (DNS) was designed by Mockapetris in 1984 as a replacement for the manually maintained and shared HOSTS.TXT file used in Internet Protocol (IP) networks to map hostnames to IP addresses, which was becoming increasingly impractical as networks grew in size and complexity [14, 15]. Instead, DNS offers a naming system that associates hierarchical alphanumeric identifiers, referred to as domain names, with various resource records, like IP addresses. In this context, a zone is defined as the set containing a domain and all of its subdomains.

Citing significant scalability concerns due to the expected size of the service and the frequency of resource record updates, Mockapetris listed the distributed storage and management of domain name entries with local caching as a design goal for DNS. To address this, the naming system information is distributed as zones amongst many name servers such that each name server is capable of either directly operating on the requested domain name resource records or referring to another name server which is hierarchically closer to the requested domain name. The name server which manages a zone is considered the authoritative name server for the zone. With this, DNS can be used to translate from the more flexible and easily remembered domain names into the associated IP addresses and other resources required to access network applications and services.

2.2.1 Name Resolution Process

A client attempting to resolve a domain name may invoke requests across several name servers. Typically, the client operates as a stub resolver which delegates the task to a known recursive resolver, such as through their network router or Internet service provider (ISP). This is generally done to allow for the caching of DNS query results for use by a whole network or organisation. This situation can be seen in Fig. 2.2.1, where the client has requested the recursive resolver to retrieve resource records for ‘www.example.com’. To execute the DNS query, the recursive resolver needs to locate the authoritative name server for the requested domain name. Without prior knowledge or cached results, the resolver first queries one of the well-known root name servers to begin navigation of the name hierarchy. In accordance with the distributed nature of DNS, the root name server does not contain the resource records for the requested domain name, but instead directs the resolver to the top-level domain (TLD) name server for ‘.com’. The resolver reiterates its query to the TLD name server and is again pointed further down the hierarchy, this time to the authoritative name server for ‘example.com’. Finally, the resolver queries this name server and retrieves the request resource records, which are then returns to the client. All of these results are cached by the recursive resolver for a set amounts of time as specified by the Time To Live (TTL) contained within all responses to help reduce overall load on the system, especially root name servers.

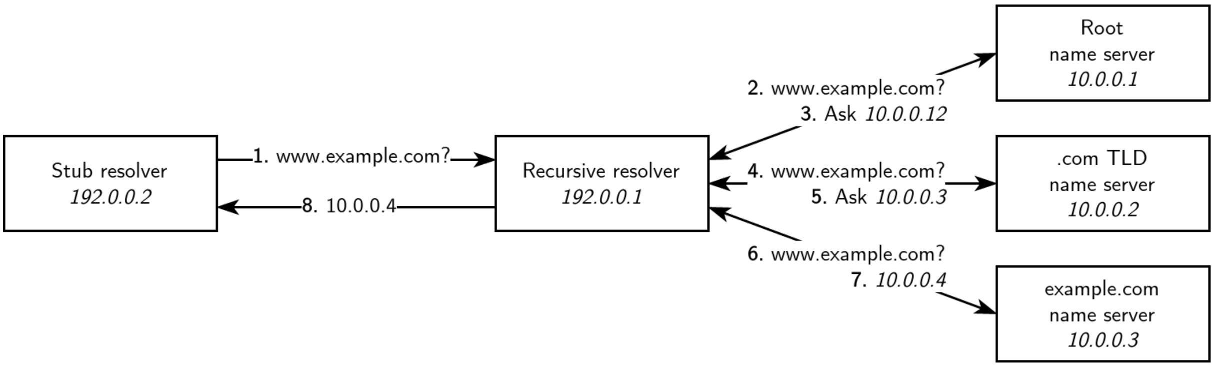


Figure 2.2.1: A stub resolver requests a recursive resolver to retrieve resource records for ‘www.example.com’. Without previously cached query responses, the recursive resolver must navigate the domain name hierarchy starting at a known root name server.

2.2.2 DNS over HTTPS

Notably, Mockapetris makes no mention of security nor privacy in the original DNS specification and such concerns have only begun to be addressed in recent years, as summarised by Bortzmeyer in 2015 [16]. This has largely been due the naming system information being perceived as public knowledge and not requiring security mechanisms. As such, DNS query and response communication have historically been sent unencrypted using the User Datagram Protocol (UDP). It has not been until the last decade with the revelations of widespread global surveillance that issues such as these have started to see much more attention. In 2013, Cooper et al. wrote extensively on the formulation of privacy threats and mitigations for consideration during the design of Internet protocols, and lists surveillance as being a prevalent privacy threat [17]. Following this, Farrell and Tschofenig emphasised the danger of exposing protocol content and metadata to large scale surveillance operations and recommend mitigation through security-conscience protocol design [18].

In an effort to apply these learnings, both DNS over TLS (DoT) and DNS over HTTPS (DoH) were conceived as methods for performing privacy-preserving DNS queries [19, 20]. Both protocols add confidentiality and data integrity to DNS by encapsulating queries and responses inside secure TLS channels. The most notable difference between the standards is the port number used, as DoT traffic goes to the non-standard port 853 while DoH is served through the standard HTTPS port 443. This difference has led to some adoption problems with DoT when compared to DoH, as it is not unusual for network firewalls to prohibit traffic to non-standard ports. This also has the effect of making DoT usage being quite conspicuous, while DoH disguises itself amongst other HTTPS traffic. García et el. list these as factors when measuring a wider adoption of DoH in 2021 [21].

2.2.3 The HTTPS Resource Record

Today, a number of DNS resource record types exist to fulfil more complex requirements and introduce advanced capabilities. The HTTPS resource record and the more general Service Binding (SVCB) resource record have recently been standardised to allow for specification of additional parameters related to service endpoint discovery and connection establishment [22]. This enables more information to be provided to the client needed to access a service while helping to avoid unnecessary round trips and DNS queries. This information set can include items such as the preferable IP address, port number and ALPN list used to connect to a service endpoint, which must otherwise be retrieved separately through potentially suboptimal channels.

While ostensibly useful for reducing overall connection latency, the ability for these new resource records to associate parameters with service endpoints facilitates much more flexibility within DNS. In particular, the ECH extension delegates public key and metadata dissemination to this mechanism though the specification of an appropriate ‘ech‘ parameter for each service endpoint.

2.3 Encrypted Client Hello

Encrypted Client Hello (ECH) is a proposed extension to TLS 1.3 which has begun to see implementation and adoption on the Internet [1–3]. ECH seeks to allow encryption of the ClientHello message, which can contain potentially sensitive information such as the SNI and ALPN extensions. Exposure of the target domain name of the client's request through the SNI was previously considered acceptable due this information being revealed through other channels, but these leaks are becoming less exploitable: Cloud hosting providers, content delivery networks (CDNs) and reverse proxies have diluted the mapping from IP addresses to domain names, the use of encrypted DNS such as DoH is now concealing client DNS queries and the TCP 1.3 handshake encrypts the server certificate. As we have seen in the previous sections, the TLS and DNS ecosystems have adapted to new security and privacy expectations in recent years and are now equipped to support ECH.

The functionality of ECH is based on clients using the public key of an ECH-service provider to send an encrypted TLS 1.3 ClientHello message, which the provider decrypts and uses to proxy the TLS 1.3 connection to the true origin server. This provider may be common to many origin servers hosting many private domains that together form an anonymity set. The provider must first generate an ECH encryption key pair and some associated metadata. This public key and metadata, referred to as an ECH configuration or ECHConfig, may then be shared out-of-band with ECH-enabled clients though a secure context like DoH using the 'ech' parameter in HTTPS resource records. A client may then use this public key and metadata to construct a ClientHello message, named the ClientHelloOuter, holding unremarkable values for the provider alongside the ECH extension containing an encrypted ClientHello, named the ClientHelloInner, itself holding the real values for a private domain. To establish a TLS connection to the origin server of this domain, the client initiates a TLS connection using the ClientHelloOuter with the provider, which decrypts the ClientHelloInner and relays the connection to the origin server, which itself completes the TLS handshake with the client through the provider. Importantly, the provider is incapable of eavesdropping on this secure channel, as the TLS connection is authenticated and end-to-end encrypted between the client and origin server.

2.3.1 Hybrid Public Key Encryption

ECH uses the Hybrid Public Key Encryption (HPKE) specification for performing public key encryption [23]. HPKE defines a standard scheme for combining the benefits of asymmetric and symmetric cryptographic algorithms, such that the performance of symmetric cryptography can be gained where only the public key of the receiver is known. This is achieved through using the public key of the receiver to generate a symmetric encryption key as well as an encapsulated shared secret. This encapsulated shared secret can be sent to the receiver, which can generate the symmetric encryption key using its private key. Any ciphertext produced by the sender with the symmetric encryption key can now be decrypted by the receiver.

HPKE defines several possible configurations of cryptographic parameters, namely selecting the key encapsulation mechanism (KEM), key derivation function (KDF) and Authenticated Encryption with Associated Data (AEAD) symmetric encryption algorithm. In ECH, these are defined to be elliptic-curve Diffie–Hellman (ECDH) using Curve25519, hashed message authentication code (HMAC) KDF (HKDF) and Advanced Encryption Standard (AES) in

Galois/Counter Mode with 128-bit key sizes (AES-128-GCM), respectively. The KEM and KDF are able to produce the AES key and encapsulated shared secret from the contents of an ECHConfig generated by the ECH-service provider. AES encrypts the ClientHelloInner and ensures the ClientHelloOuter can not be tampered using its additional authenticated data (AAD) mechanism. Once the ClientHelloOuter containing the ECH extension is received, the provider can derive the same AEAD key from the encapsulated shared secret using the KDF with its private key and then decrypt the ClientHelloInner. Bhargavan, Cheval, and Wood have been able to verify the security of HPKE in the context of ECH through extensive formal analysis of the privacy properties of the TLS 1.3 handshake [24].

2.3.2 Split Mode Deployment

The ECH protocol is designed to operate within two types of network topologies, referred to as Shared Mode and Split Mode. When in Shared Mode, the ECH-service provider and private domain origin server are the same network entity. The TLS 1.3 connection initiated by ECH-enabled clients with the provider is also completed by the provider, which can then serve the client the covertly requested domain name service. Split Mode relaxes this restriction to allow the physical separation of the provider and origin server. An example execution of ECH in Split Mode is visualised in Fig. 2.3.1, where we see the client again initiates a TLS 1.3 connection with the provider, but this connection is forwarded to the appropriate origin server which completes the connection. In either case, the true ClientHello message is still masked from network observers and the anonymity set consists of all possible private domains served via the provider using ECH Shared Mode or Split Mode.

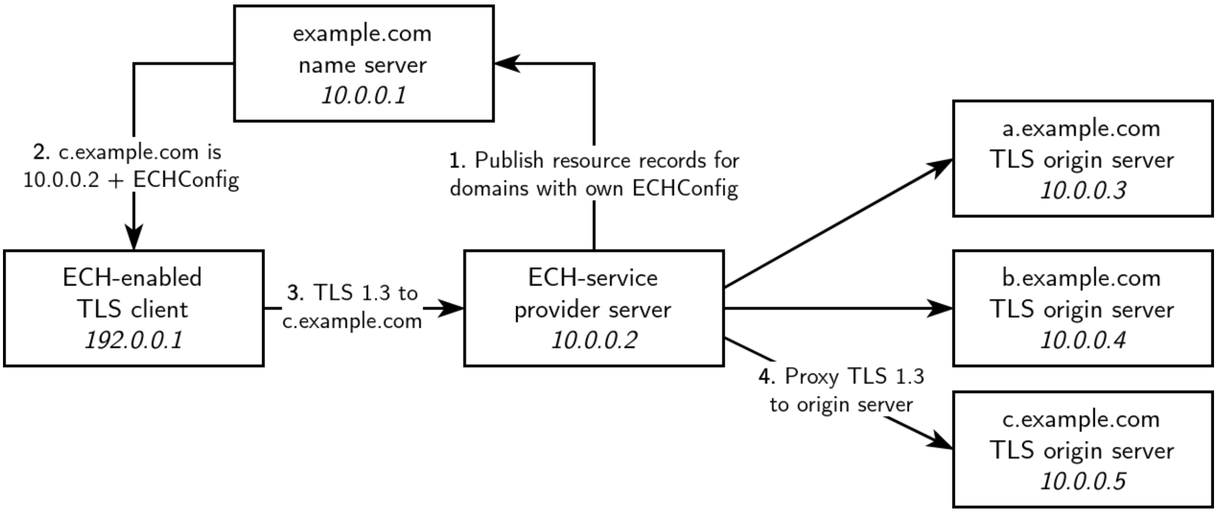


Figure 2.3.1: An example of how ECH can be used in Split Mode when the ECH-service provider is not co-located with the requested private domain origin server. Steps 1 and 2 are completed out-of-band. Steps 3 and 4 result in a TLS connection being established between the client and origin server.

Split Mode is crucial for unbinding the provider from the origin server which is necessary for being able to handle more diverse ECH deployment scenarios. This is generally required when the provider does not have the resources to complete the TLS connection and must proxy the connection to the requested origin server. This can be the case in cloud hosting environments, where the private keys and certificates of hosted services belong to the customer and cannot be accessed by the reverse proxies operated in front of the internal cloud network infrastructure. However, if the client, provider and origin server are separated by a public network such that traffic on both the client to provider and provider to origin server channels can be intercepted by a foreign network observer, it is not enough to encrypt this traffic with TLS 1.3 to prevent the observer from learning which origin server the client is interacting with, which may eliminate any privacy offered to the client by the provider's anonymity set. This is possible through an attack accomplished using traffic analysis.

2.4 Traffic Analysis

Traffic analysis is the process of passively recording and inspecting possibly large amounts of messages sent over a network in order to discern information not apparent when considering each message in isolation. Traffic analysis techniques can be used to infer sensitive information from patterns in network activity regardless of channel encryption, as they exploit fundamental aspects inherent to how a communication system is implemented. For example, consider that the mere presence of network traffic between a household and a specific medical, educational or political institution's web server might tell us a great deal about the lifestyle and affiliations of the occupants without needing to know anything about the contents of the traffic itself. Aside from analysing traffic behaviour, other techniques include inspecting network protocols being used, observing changes in round-trip latency and comparing packet sizes and contents.

2.4.1 Traffic Correlation Attacks

Traffic correlation attacks describe a large subset of traffic analysis techniques identified by their use of correlating patterns found in network channels to detect associations between entities that were otherwise not evident. These can typically be used to unmask users and their activities in anonymisation networks. Back, Möller, and Stiglic have reported on correlation metrics such as packet counting and traffic shaping employed against the Freedom network while DeFabbia-Kane has additionally found packet timing and inter-packet delay to be effective against Tor [25, 26]. Most significantly for this paper, Trevisan et al. have shown ECH operating over a public network is highly susceptible to traffic correlation attacks using a conventional machine learning algorithm trained on information extracted from the IP, TCP and UDP protocol fields, TLS SNI values, packet sizes and inter-packet delays [27].

2.4.2 Countermeasures

The effectiveness of traffic correlation attacks can be mitigated by disrupting the recognisable patterns in communication through removing distinctive features and inserting randomness into messages and traffic flow. Back, Möller, and Stiglic saw how PipeNet introduces dummy packets into the network between correspondents as traffic padding and uses mixing and pacing with a packet scheduling algorithm at each node to hinder attack vectors.

This is a particularly hard challenge for low-latency network systems such as web servers and instant-messaging platforms because many mitigations require the introduction of unacceptable delays or continuous high bandwidth usage. Levine et al. conducted a study on using packet timing analysis to attack low-latency anonymisation networks and concluded the effectiveness of traffic padding can be improved by intentionally occasionally dropping dummy packets [28]. Wright, Coull and Monrose have suggested morphing classes of encrypted traffic into indistinguishable distributions with a mathematical model for minimising differing features over time [29].

2.5 Summary

TLS and DNS continue to evolve as their requirements shift in response to modern security and privacy demands. From this movement, the ECH extension for TLS 1.3 has emerged to enable the encryption of the ClientHello message and thereby addressing one of the last points an attacker can learn of potentially sensitive information, such as the SNI and ALPN list. The ECH standard defines Split Mode as a network topology which permits the ECH-service provider to be physically separate from the origin server. However, such a situation reveals a potential attack surface against the extension through traffic correlation, which must be disrupted using various practical countermeasures.

3 Design

This chapter defines the challenges associated with the research problem and presents a refined solution. We then delve deeper into the individual components to explore the reasoning and considerations that produced this design, and outline the benefits and limitations of alternative approaches. The chapter aims to showcase this idea independent of implementation, instead using generalised concepts that may be applied by the reader within their own practice.

3.1 Problem Overview

As we saw in Chapter 1, a distributed deployment model for ECH exhibits some desirable qualities but also constitutes quite a complicated problem to solve. There are many factors at play, particularly around orchestrating TLS server co-operation, publishing ECHConfig values associated with specific servers, instructing TLS client behaviour and impeding traffic analysis attacks. Furthermore, these have to function within the confines of our imperfect world where we must accept headaches of technological inertia and legacy systems. This has only grown in prevalence on the Internet, with backbone technologies like IPv4, DNS and HTTP requiring countless workarounds or extensions to meet our modern demands without breaking backwards-compatibility.

The overall form of the solution appears somewhat similar to how conventional ECH in Split Mode operates. In Section 2.3.2, it was highlighted how Split Mode provides a means to separate ECH-service providers from private domain origin servers, but makes no attempt to remain secure when deployed across a public network nor facilitate a multi-provider setup. The approach taken here differs primarily in that the DNS resource records for a private domain can point to any IP address in the co-operative network while all co-operating TLS servers are linked together over public channels. We can see such a scenario depicted in Fig. 3.1.1, where when examining Split Mode previously we saw three origin servers accessible through a single provider, we can now envision a client capable of querying any of the three origin servers for a private domain and having their connection transparently reach through to the correct origin server.

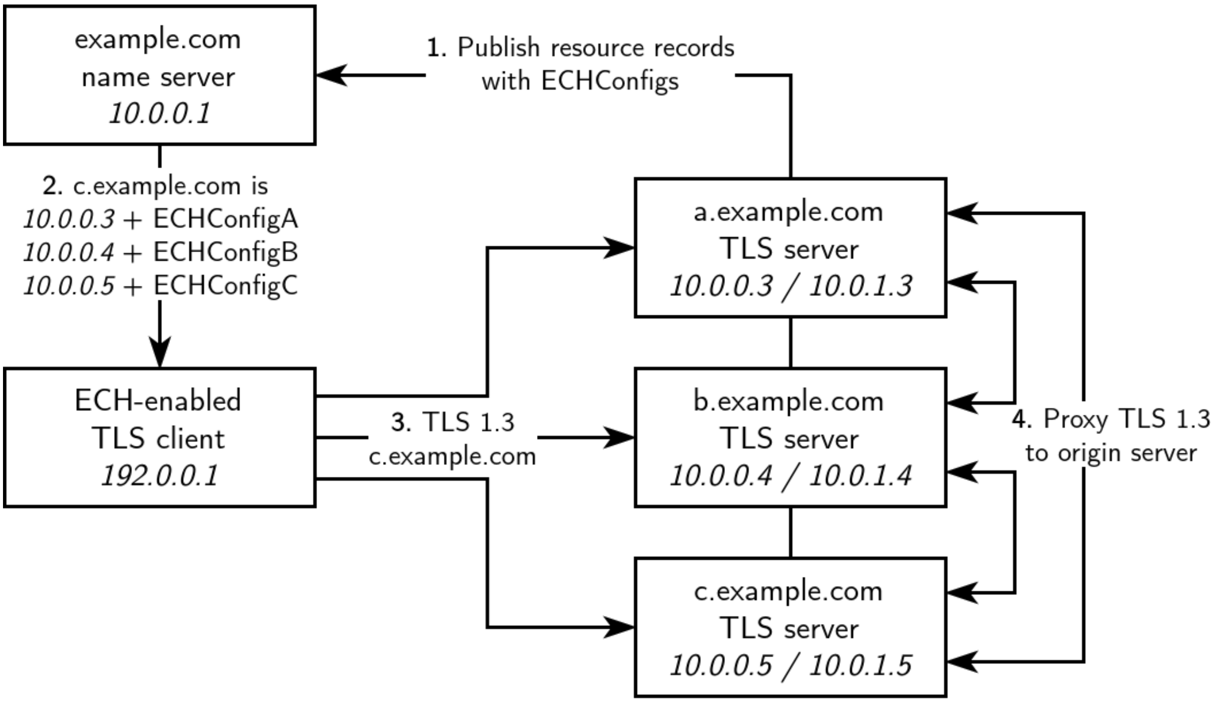


Figure 3.1.1: An example distributed deployment of ECH using the proposed solution. Steps 1 and 2 are completed out-of-band while steps 3 and 4 result in a TLS connection being established between the client and appropriate origin server through another TLS server.

This diagram hides some technical details and considerations that should be kept in mind. This loose network of TLS servers somehow need to publish DNS resource records for all private domains that can point at any one of them while referencing the corresponding ECHConfig value. Additionally, one design criteria is for load to be allocated across the network in a reasonably fair manner, so it is expected that every server is continuously operating as an ECH-service provider for other servers. As such, it is not enough to regularly cycle through sets of DNS resource records pointing towards each server one at a time but we must instead find a method to have the same DNS query uniformly spread clients amongst participating servers. This is especially hard if we want all servers to have distinct private keys and thus different ECHConfig values, as now we need to somehow associate this value with an individual server rather than with the domain name, which is a historically difficult challenge in DNS.

As both the client-server and server-server channels are over public networks, this reveals a perfect attack surface for any network observers to perform a correlation attack between server ingress and egress traffic. Even after an origin server completes the TLS handshake and establishes end-to-end encrypted communication with a client, an eavesdropper can deduce which origin server the client is interacting with by comparing ingress/egress times and packet counts.

To address these challenges, the design has been split into two topics of interest: Identifying mechanisms needed for enabling distributed deployment and disrupting correlation attacks using traffic obfuscation techniques.

3.2 Distribution Mechanisms

The previous section raised a number of criteria that must be fulfilled to allow for this distributed deployment model, which can be summarised into two points. Firstly, the solution requires a series of procedures that enables the use of DNS resource records to direct ECH-enabled clients evenly between all co-operating TLS servers with correct ECHConfig values when resolving private domain names using encrypted DNS. Then, these servers need to be able to forward client connections to each other over a public network without exposing the decrypted ClientHelloInner.

3.2.1 DNS Publication Schema

We can take advantage of the commonly seen round-robin DNS technique to share load throughout the co-operative network. Round-robin DNS works by responding to DNS queries with multiple valid IP addresses in a random order from which the client selects one. This is typically used to provide simplistic uniform load distribution without the need for dedicated load balancing software or hardware. By installing the IP addresses of all co-operating TLS servers into all private domain name DNS resource records, clients who resolve any of these domains will be directed any one of the servers. This has the additional benefit of allowing clients to immediately retry a connection with the next IP if the connection fails, although some clients might not implement this. However, a major flaw with this approach is requiring all servers to share an ECH private key because there is no way to specify which ECHConfig value should be used by the client when it selects an IP address. As such, this mechanism can only work if there is exactly one ECHConfig value to choose from.

If we are not limited to static declaration of DNS resource records, a better method can be employed here using a dynamic DNS service. We use software to regularly substitute private domain name DNS resource records such that load is fairly balanced across servers. This allows us to specify specific IP address and ECHConfig value pairs for each domain name, alleviating the need for sharing private keys. Such software can also make intelligent decisions based on real-time information such as amount of DNS queries or even reported TLS server traffic flow, which could greatly improve the fairness offered by the load balancing system in cases of heterogeneous server capabilities. Unfortunately, using dynamic DNS for load balancing is prone to sporadic inconsistency due to DNS response caching. Not only can stale cache entries send traffic to wrong locations, this can suddenly result in more traffic appearing on one IP address if a busy recursive resolver chooses to cache a response for many stub resolvers to use. Lowering the Time-To-Live (TTL) of DNS resource records may help, at the cost of more frequent polling.

It may also be worthwhile to investigate HTTPS resource records “alternative endpoint” functionality, which may be able to associate ECHConfig values with individual servers rather than domain names. However, this project was not able to get consistent results across different platforms with these.

3.2.2 TLS Server Co-operation

Due to servers operating as both an own origin server for their own domains and as an ECH-service provider for all other domains in the anonymity set, they need to accept both regular ClientHello messages and ClientHello messages containing the ECH extension with an encrypted ClientHelloInner. While a regular ClientHello message results in the usual TLS handshake, after decryption a ClientHelloInner can lead to either ECH in Shared Mode or Split Mode. In Shared Mode, the ClientHelloInner can now be processed as a regular ClientHello message, but Split Mode requires the server proxy the TLS connection the appropriate origin server.

When forwarding this connection to the origin server, the ClientHelloInner is now sent as a regular ClientHello without any security, entirely defeating the purpose of ECH. To solve this issue, server-server communication have some form on encryption to maintain the confidentiality of the ClientHelloInner. As widely shared secrets should be avoided and we would like this co-operation network to remain adaptable to change, a KEM and KDF backed by public key cryptography should be considered to establish ephemeral shared secrets between servers for conducting symmetric encryption.

3.3 Traffic Obfuscation

In addition to protecting the ClientHelloInner from exposure, this design also seeks to prevent revelation of which origin server a client is communicating with, as such knowledge could significantly reduce or eliminate the anonymity set cloaking the target domain name. We have already seen in Section 2.4.1 that ECH is susceptible to traffic correlation attacks through machine learning classifier models and that there exists many metrics useful for traffic pattern recognition. Then in the following section, it was noted that low-latency network systems are especially vulnerable to timing analysis attacks. Together, these present a difficult challenge for ensuring the security of ECH when using distributed deployment.

In an attempt to mitigate these attacks, this design introduces obfuscation on top of encryption into all server-server channel traffic. These techniques are based on the goal of minimising features and interrupting patterns that occur in communication systems that can

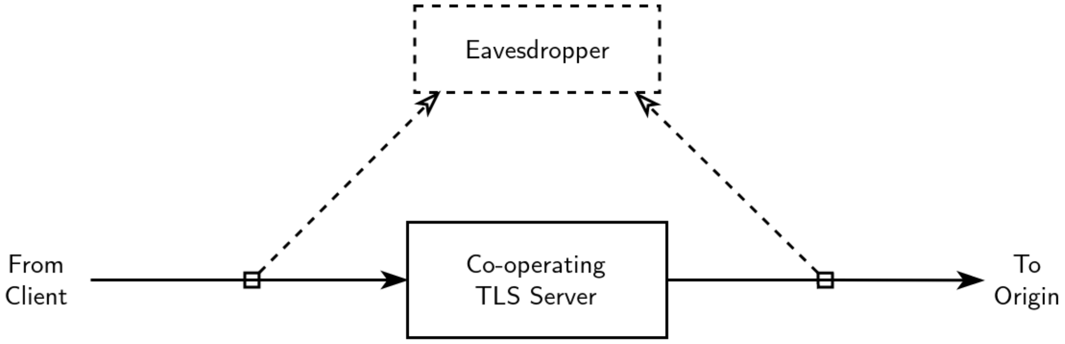


Figure 3.3.1: A network observer with view a of client-server and server-server channels has the opportunity to launch correlation attacks.

be used to find associations between channels. As seen in Fig. 3.3.1, if an eavesdropper can record both the ingress and egress traffic of a co-operating server, they may be able to track the movement of packets from one channel to another due to these conspicuous traits. Such a situation is not improbable, as in reality both channels are likely to pass through the same physical infrastructure or organisation at some point.

3.3.1 Normalisation

Normalisation is the process of morphing arbitrary traffic network such that a given metric remains a fixed value over time. For example, instantaneous throughput could be considered an aspect of traffic that can be correlated with another the activity in another channel. Normalisation can apply traffic shaping rules and inject noise to ensure their is a constant amount of throughout at any one time. Normalisation for masking the absence of traffic can be somewhat impractical for use in civilian-settings as bandwidth can be both limited and variable, so it might not be an economic viable approach.

3.3.2 Pacing and Mixing

While not perfect, many practical techniques exist to mask traffic with minimal impact to overall network performance. For instance, the lengths of packets may be rounded to the next multiple of 32 or such that it blends in with surrounding activity [30]. To disrupt timing-based correlation attacks, delays and packet slotting can be used to pace the rate of traffic. Traffic mixing, where dummy traffic is introduced into the stream, has also proven to be effective when used in conjunction with variable inter-arrival times [31, 32].

3.4 Summary

Every co-operating TLS server operates both as ECH-service provider for all other members and as an origin server for any domains it serves. Round-robin static DNS and dynamic DNS can be used with the HTTPS resource record to fairly distribute ECH-enabled client traffic amongst these servers, but the round-robin technique requires all servers to shared a private key. Without intervention, passive observers would be able to read ClientHelloInner values from server-server channels and undermine anonymity sets through traffic correlation. To solve this, server-server communication is encrypted and further obfuscated using traffic pacing and dummy packet mixing.

4 Implementation

In order to evaluate the effectiveness of the solution presented in Chapter 3, an implementation is developed using commonplace software and tools. This chapter outlines the steps taken to simulate a working prototype of the design in a virtual environment where metrics can be recorded inside reproducible scenarios. The project work produced as a result of this effort has been shared for your convenience and can be found in Appendix A1.

In this chapter, the process for establishing a deterministic, configurable and measurable network evaluation environment using DebVM and QEMU is demonstrated. This is followed by an explanation and discussion of how BIND9 and NGINX are configured to realise the distributed deployment of ECH. Additionally, various traffic obfuscation techniques are instated using a combination of wireguard, tc and tcpdump. Finally, curl, Mozilla Firefox and Google Chrome are configured with ECH-support and shown to work with distributed ECH deployment.

4.1 Simulation

The initial time investment placed into a script for quick and configurable virtual testing environments was paramount for saving significant time and stress during the later iterative development process used to refine the design of the solution. The ability to teardown and setup a fresh virtual environment containing the exact same configuration in under a minute was vital during periods of investigation and debugging as well as evaluation and data collection. These virtual environments consist of many virtual machines connected together using a bridge network device.

4.1.1 Virtualisation

The virtual machines are created using a tool called DebVM, which is a composition of QEMU and mmdebstrap. QEMU is an application and hardware emulator that manage the execution of virtual machines while mmdebstrap is a tool to create Debian Linux operating systems [33]. DebVM is a thin abstraction to these and offers a user-friendly interface to create and run new instances of Debian Linux.

This project requires building OpenSSL, curl and NGINX from source patched with ECH support as well as the configuration of several virtual machines. This can be a very lengthy process, so effort was made to separate these steps into composable QEMU images that together form the complete virtual machine. An excerpt of this process has been included in Listing. 4.1.1. The configuration of these images are automated using non-interactive SSH commands. Seen here is the assembly of a QEMU image called build.img that contains the output of the build process. Primarily due to storage restrictions, it was decided to isolate the virtual machine which contains all the necessary build tools from the composed virtual machines.

```
ssh-keygen -N "" -t ed25519 -f ssh.key
debvm-create -h builder -o builder.img -r unstable -z 2GB -k ssh.key.pub -- \
    --include ca-certificates,build-essential,dh-autoreconf,git,e2fsprogs \
    --include libpsl-dev,libpcre3-dev,libz-dev,libnghttp2-dev

qemu-img create build.img 2G
debvm-run --image builder.img --sshport 2222 --graphical -- \
    -display none -drive file=build.img,format=raw,if=virtio,readonly=off &
debvm-waitssh 2222

ssh -o NoHostAuthenticationForLocalhost=yes -i ssh.key -p 2222 root@127.0.0.1 "
    mkfs.ext4 -L build /dev/vdb
    mount /dev/vdb /mnt

    git clone -b ECH-draft-13c https://github.com/sftcd/openssl.git /mnt/src/openssl
    cd /mnt/src/openssl
    ./config --prefix=/mnt/openssl --openssldir=/mnt/openssl
    make -j8
    make -j8 install

    cd / && umount /mnt
    shutdown now"
wait
```

Listing 4.1.1: Building OpenSSL from source inside build.img with DebVM.

In a similar procedure, the software and configuration common to all virtual machines are built into a base image. New virtual machines are initialised as a snapshot of the base image with further configuration applied on top. To boot up these virtual machines, QEMU instances of them are spawned into the background in parallel. Each virtual machine exposes one SSH port bound to a unique port of the host machine to allow for interactive control and customisation.

4.1.2 Networking

A bridge network device is created and managed on the host machine to allow packets to travel between virtual machines. DebVM allows for specifying additional arguments that are passed to QEMU, and can be seen used to connect a virtual machine to a bridge network in Listing 4.1.2.

```
sudo ip link add name br0 type bridge
sudo ip addr add 172.0.0.1/24 dev br0
sudo ip link set dev br0 up

debvm-run --image host.img -- \
    -device virtio-net-pci,netdev=net1,mac=00:00:00:00:00:01 \
    -netdev bridge,id=net1,br=br0
```

Listing 4.1.2: Connecting QEMU virtual machines together using a network bridge.

Each virtual machine has a static network configuration with a unique IP and MAC address that is installed to /etc/systemd/network/00-br0.network during initialisation, which is automatically applied by systemd early after boot up. An example configuration file is provided in Listing 4.1.3.

```
[Match]
MACAddress=00:00:00:00:00:01

[Network]
DNS=172.0.0.254
Address=172.0.0.5/24

[Route]
Gateway=0.0.0.0
Destination=0.0.0.0/0
Metric=9999
```

Listing 4.1.3: Static bridge network configuration using systemd.

As previously described in Section 2.1.1, digital certificates provide an authentication service through a chain of trust model. Well-behaved TLS entities on the network will only interact with other TLS entities that possess a certificates that have been signed by a trusted CA. To shortcut the process of applying for an official certificate trusted CA, we can instead create our own root CA with a self-signed certificate using OpenSSL. Listing 4.1.4 uses the patched version of OpenSSL that was built in the previous section.

```
LD_LIBRARY_PATH=/mnt/openssl/lib64 /mnt/openssl/bin/openssl req -x509 \
    -newkey ec -pkeyopt ec_paramgen_curve:secp384r1 -days 3650 -nodes \
    -keyout /keys/root.key -out /keys/root.crt -subj '/CN=example.com'
```

Listing 4.1.4: Generating a new self-signed root CA X.509 certificate using OpenSSL.

New certificates can now be issued by the root CA as seen in Listing 4.1.5. However, no TLS entity will be able to establish trust with these certificates as it has not been signed by a trusted CA. We will see in Section 4.4 how applications can be configured to trust new root CAs and thus trust these signed certificates.

```
LD_LIBRARY_PATH=/mnt/openssl/lib64 /mnt/openssl/bin/openssl req \
    -newkey ec -pkeyopt ec_paramgen_curve:secp384r1 -nodes \
    -keyout /keys/dns.key -out /keys/dns.csr -subj '/CN=ns.example.com'

LD_LIBRARY_PATH=/mnt/openssl/lib64 /mnt/openssl/bin/openssl x509 -req \
    -CA /keys/root.crt -CAkey /keys/root.key -days 3650 -CAcreateserial \
    -extfile <(printf 'subjectAltName=DNS:ns.example.com') \
    -in /keys/dns.csr -out /keys/dns.crt
```

Listing 4.1.5: Signing a new X.509 certificate for ns.example.com using OpenSSL.

With these steps, we have the foundations for building a virtual environment on top of. The rest of this chapter is comprised of the steps taken to implement the different servers, clients and applications that make up the virtual environment.

4.2 DNS Server

One of the virtual machines operates as the DNS server for clients to query. The Berkeley Internet Name Domain version 9 (BIND9) software provides all the functionality needed for this prototype [34]. Currently, all practical ECH-enabled clients require DoH to be used when resolving the private domain name for ECH to be performed. To enable this in BIND9, we provide a private key and signed certificate for use by TLS and then begin listening on port 443 as seen in Listing 4.2.1.

```
tls tlspair {
    key-file "/keys/dns.key";
    cert-file "/keys/dns.crt";
};

options {
    directory "/var/cache/bind";
    recursion no;
    dnssec-validation auto;
    allow-transfer { none; };
    listen-on { any; };
    listen-on port 443 tls tlspair http default { any; };
};

zone "example.com" {
    type master;
    update-policy local;
    file "/var/lib/bind/db.example.com";
};
```

Listing 4.2.1: DNS over HTTPS configuration using BIND9.

BIND9 uses zone files to specify naming system information. The round-robin static DNS technique has been implemented Listing 4.2.2. Note that <Shared_ECHConfig> would be replaced with the base64-encoded ECHConfig value that would be common between all co-operating TLS servers.

```
$ORIGIN example.com.
$TTL 3600

@ IN SOA dns root.dns 2024040100 3600 600 86400 600
@ IN NS dns

dcu IN A 172.0.0.2
dcu IN A 172.0.0.5
dcu IN A 172.0.0.8
dcu IN HTTPS 1 . ech=<Shared_ECHConfig>

tcd IN A 172.0.0.2
tcd IN A 172.0.0.5
tcd IN A 172.0.0.8
tcd IN HTTPS 1 . ech=<Shared_ECHConfig>

ucd IN A 172.0.0.2
ucd IN A 172.0.0.5
ucd IN A 172.0.0.8
ucd IN HTTPS 1 . ech=<Shared_ECHConfig>
```

Listing 4.2.2: example.com zone file for distributed ECH using a shared ECH key.

As discussed in Chapter 3, this requires all servers to share a private key. In order to avoid such widely shared secrets, the dynamic DNS approach can be employed to regularly update the DNS resource records. The Listing 4.2.3 uses the nsupdate tool to send DNS zone updates using the DNS UPDATE specification [35]. Notice we must also enable dynamic DNS in BIND9 as seen in Listing 4.2.1 Line 17.

```
pairs="172.0.0.2,<DCU_ECHConfig> 172.0.0.5,<TCD_ECHConfig> 172.0.0.8,<UCD_ECHConfig>"
while true; do
    dcu=$(shuf -e $pairs); tcd=$(shuf -e $pairs); ucd=$(shuf -e $pairs); echo "
    update delete dcu.example.com
    update add dcu.example.com 60 A ${dcu%%,*}
    update add dcu.example.com 60 HTTPS 1 . ech=${dcu#*,}
    update delete tcd.example.com
    update add tcd.example.com 60 A ${tcd%%,*}
    update add tcd.example.com 60 HTTPS 1 . ech=${tcd#*,}
    update delete ucd.example.com
    update add ucd.example.com 60 A ${ucd%%,*}
    update add ucd.example.com 60 HTTPS 1 . ech=${ucd#*,}
    send" | nsupdate -l
    sleep 1
done
```

Listing 4.2.3: Rudimentary script to implement a dynamic DNS service.

This script selects a random server IP and ECHConfig pair to assign to each private domain name DNS resource records. Note that `<DCU_ECHConfig>`, `<TCD_ECHConfig>` and `<UCD_ECHConfig>` would be replaced with the base64-encoded ECHConfig value belonging to the appropriate TLS server.

4.3 TLS Server

The remaining virtual machines in the environment function as the co-operating TLS servers. Similar to the DNS server in Listing 4.1.5, TLS server are issued signed certificates. However, they receive one certificate for every domain they serve. Additionally, if using the dynamic DNS service method, each server needs its own ECH key pair, which can be generated using the command in Listing 4.3.1. It is from this key pair that the values for `<xxx_ECHConfig>` can be found.

```
LD_LIBRARY_PATH=/mnt/openssl/lib64 /mnt/openssl/bin/openssl ech \
    -public_name tcd.example.com -pemout /keys/tcd/key.ech
```

Listing 4.3.1: Generating a new ECH key pair for tcd.example.com using OpenSSL.

The patched version of NGINX built earlier is used as the web server. Listing 4.3.2 provides a configuration which implements the previously described design.

```
stream {
    ssl_preread on;
    ssl_echkeydir /keys/tcd;
    server { listen 172.0.0.5:443; proxy_pass $origin; }
    map $ssl_preread_server_name $origin {
        dcu.example.com 172.0.1.2:443;
        tcd.example.com 172.0.1.5:443;
        ucd.example.com 172.0.1.8:443;
    }
}

http {
    server {
        root /site/tcd;
        server_name tcd.example.com;
        listen 172.0.1.5:443 ssl;
        http2 on;
        ssl_certificate /keys/tcd/tcd.crt;
        ssl_certificate_key /keys/tcd/tcd.key;
        ssl_protocols TLSv1.3;
        location / { ssi on; index index.html; }
    }
}
```

Listing 4.3.2: Distributed ECH NGINX configuration for tcd.example.com.

It functions by listening on two interfaces: All connections towards 172.0.1.5:443, the server-server interface, are processed as regular ClientHello messages while connections towards 172.0.0.5:443, the public interface, are processed as ClientHelloOuter messages. Once decrypted, the ClientHelloInner is forwarded towards the appropriate origin server through the server-server interface. This interface is a virtual private network (VPN) created by WireGuard that operates over the public interface [36]. WireGuard uses a public and private key for its KEM, which are generated as shown in Listing 4.3.3. This key is then used to provide peer-to-peer symmetric encryption.

```
wg genkey | tee /keys/tcd/wg.key | wg pubkey > /keys/tcd/wg.key.pub
```

Listing 4.3.3: Generating a new WireGuard key pair for tcd.example.com.

Listing 4.3.4 is a simple traffic obfuscation script that uses the tc tool to install a packet slotting rule into the Linux networking stack [37, 38]. This forces all packets to accrue for up to a maximum time before all being sent together. Additionally, tcpdump is used to detect network activity going to a server, which results in duplicate packets being sent to all other servers. It has been noticed that traffic padding appears a short delay after real traffic. Trials have been made using socat and iptables to improve functioning. but no significant changes in effectiveness were noted. The implementation script provided in Appendix A1 continues to use socat and iptables for reference.

```
tc qdisc replace dev enp0s6 root netem slot 10ms 20ms

tcpdump -i wg0 -nnqlt udp and src 172.0.1.5 and not dst port 1234 \
        | while read _ _ _ dst _ len; do
    [ "172.0.1.2" != "${dst%.*}" ] &&
        dd if=/dev/urandom bs=$len count=1 >/dev/udp/172.0.1.2/1234 &
    [ "172.0.1.8" != "${dst%.*}" ] &&
        dd if=/dev/urandom bs=$len count=1 >/dev/udp/172.0.1.8/1234 &
done
```

Listing 4.3.4: Rudimentary script to shroud legitimate WireGuard communication.

4.4 TLS Client

This project uses three TLS clients for testing ECH functionality. These clients need to be configured to use ECH by enabling use of DoH with ns.example.com as the selected resolver and installing the root CA certificate generated in Listing 4.1.4.

4.4.1 curl

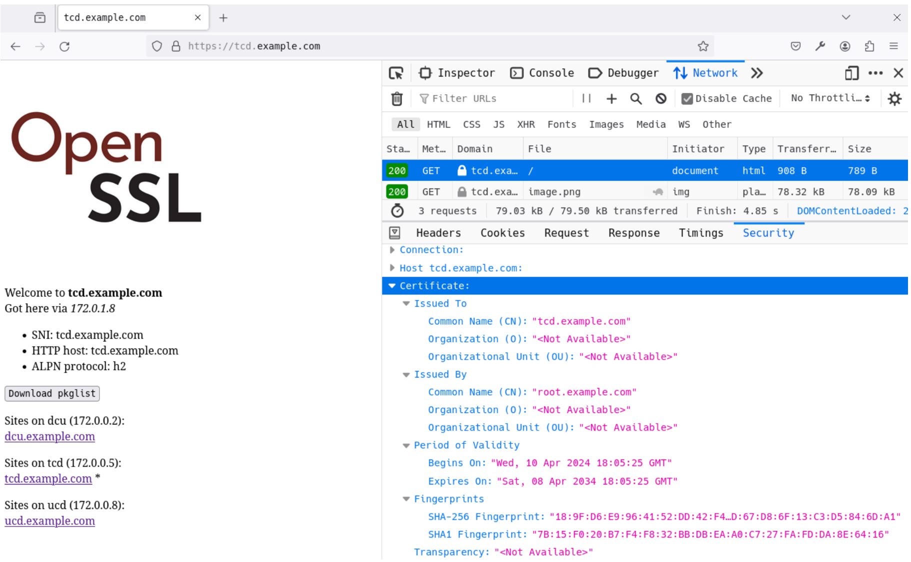
curl is a command-line client for accessing network resources. It has recently received support for ECH through the DEfO project. The project has used curl for simulating random client requests, as in Listing 4.4.1. This script selects a random private domain name and then uses curl to execute a TLS connection using ECH after resolving the domain name using DoH. A copy of this output produced by curl when querying https://tcd.example.com has in Appendix A2.

```
while true; do
    sleep 0.$(( $RANDOM % 999 ))
    LD_LIBRARY_PATH=/mnt/openssl/lib64 /mnt/curl/bin/curl \
        --verbose --cacert /keys/root.crt --ech hard \
        --doh-url https://dns.example.com/dns-query \
        https://$(shuf -n 1 -e dcu.example.com tcd.example.com ucd.example.com)
done
```

Listing 4.4.1: Command to use ECH-enabled curl on QEMU virtual machines.

4.4.2 Mozilla Firefox

Mozilla Firefox is a web browser that received support for ECH in version 118, September 2023. There were no issues encountered when configuring the browser to enable ECH support and point DoH to ns.example.com. Fig. 4.4.1 showcases the browser successfully accessing tcd.exmaple.com through 172.0.1.8, which belongs to the UCD TLS server.



Welcome to **tcd.example.com**
Got here via *172.0.1.8*

- SNI: tcd.example.com
- HTTP host: tcd.example.com
- ALPN protocol: h2

Download pkglist

Sites on dcu (172.0.0.2):
dcu.example.com

Sites on tcd (172.0.0.5):
tcd.example.com *

Sites on ucd (172.0.0.8):
ucd.example.com

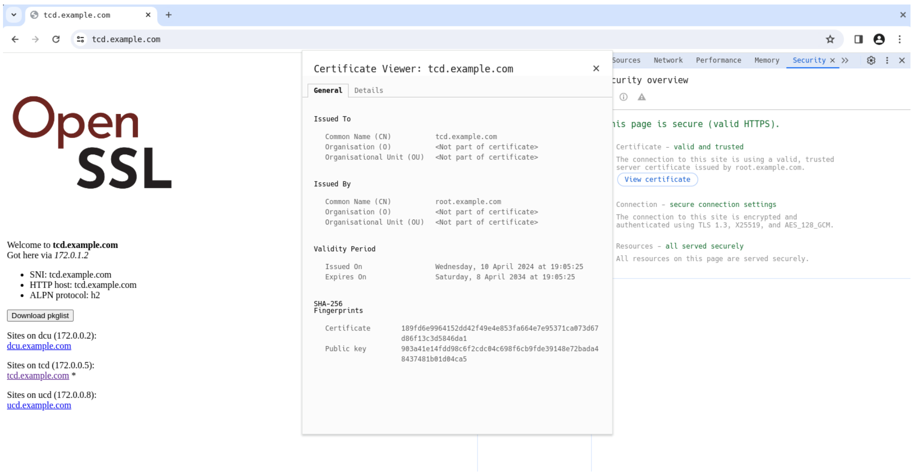
Sta…	Met…	Domain	File	Initiator	Type	Transferr…	Size
200	GET	tcd.exa…	/	document	html	908 B	789 B
200	GET	tcd.exa…	image.png	img	pla…	78.32 kB	78.09 kB

```
3 requests   79.03 kB / 79.50 kB transferred   Finish: 4.85 s   DOMContentLoaded: 2
Connection:
Host tcd.example.com:
Certificate:
  Issued To
    Common Name (CN): "tcd.example.com"
    Organization (O): "<Not Available>"
    Organizational Unit (OU): "<Not Available>"
  Issued By
    Common Name (CN): "root.example.com"
    Organization (O): "<Not Available>"
    Organizational Unit (OU): "<Not Available>"
  Period of Validity
    Begins On: "Wed, 10 Apr 2024 18:05:25 GMT"
    Expires On: "Sat, 08 Apr 2034 18:05:25 GMT"
  Fingerprints
    SHA-256 Fingerprint: "18:9F:D6:E9:96:41:52:DD:42:F4…D:67:D8:6F:13:C3:D5:84:6D:A1"
    SHA1 Fingerprint: "7B:15:F0:20:B7:F4:F8:32:BB:DB:EA:A0:C7:27:FA:FD:DA:8E:64:16"
  Transparency: "<Not Available>"
```

Figure 4.4.1: Screenshot of Mozilla Firefox when accessing tcd.example.com.

4.4.3 Google Chrome

Google Chrome is another web browser that received support for ECH in v118, October 2023. The browser was not able to use ns.example.com when set using the custom encrypted DNS option, but it was possible to work around this by setting the device's DNS resolver to ns.example.com and telling the browser to use the system resolver. Fig. 4.4.2 showcases the browser successfully accessing tcd.exmaple.com through 172.0.1.2, which belongs to the DCU TLS server.



Welcome to **tcd.example.com**
Got here via *172.0.1.2*

- SNI: tcd.example.com
- HTTP host: tcd.example.com
- ALPN protocol: h2

Download pkglist

Sites on dcu (172.0.0.2):
dcu.example.com

Sites on tcd (172.0.0.5):
tcd.example.com *

Sites on ucd (172.0.0.8):
ucd.example.com

```
Certificate Viewer: tcd.example.com

Issued To

   Common Name (CN)            tcd.example.com
   Organisation (O)            <Not part of certificate>
   Organisational Unit (OU)    <Not part of certificate>

Issued By

   Common Name (CN)            root.example.com
   Organisation (O)            <Not part of certificate>
   Organisational Unit (OU)    <Not part of certificate>

Validity Period

   Issued On                   Wednesday, 10 April 2024 at 19:05:25
   Expires On                  Saturday, 8 April 2034 at 19:05:25

SHA-256
Fingerprints

   Certificate      189fd6e9964152dd42f49e4e853fa664e7e95371ca073d67
                    d86f13c3d5846da1
   Public key       903a41e14fdd98c6f2cdc04c698f6cb9fde39148e72bada4
                    8437481b01d04ca5
```

```
curity overview

his page is secure (valid HTTPS).

Certificate - valid and trusted
The connection to this site is using a valid, trusted
server certificate issued by root.example.com.

Connection - secure connection settings
The connection to this site is encrypted and
authenticated using TLS 1.3, X25519, and AES_128_GCM.

Resources - all served securely
All resources on this page are served securely.
```

Figure 4.4.2: Screenshot of Google Chrome when accessing tcd.example.com.

4.5 Summary

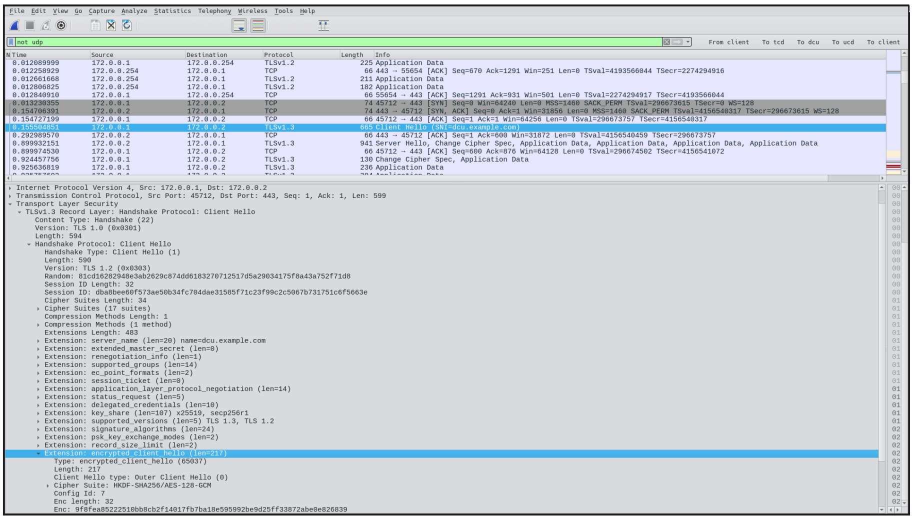
Virtual environments comprise of many virtual machines generated using DebVM and connected together over a bridge network. BIND9 provides authoritative name server functionality and NGINX is used as the web server on all co-operating TLS servers. WireGuard operates a VPN service to encrypt all server-server communication, while tc and tcpdump offer a proof-of-concept implementation of traffic obfuscation. Finally, curl, Mozilla Firefox and Google Chrome have been used as ECH-enabled clients to validate functioning of the ECH deployment model.

5 Results and Discussion

After completing an implementation of the design in Chapter 4, an evaluation of its effectiveness is conducted based on the goals of the project. In this chapter, a method for data collection using the implementation is presented, the results of the data collection are analysed, and a discussion on the overall strengths and weaknesses of the design and implementation is given. In particularly, the solutions ability to fairly balance load between co-operating TLS servers, minimising impact to overall network performance and demonstrating protection of user privacy are considered.

5.1 Data Collection

Wireshark is a tool for capturing and analysing network traffic. In this project, it is used to record all traffic passing through the bridge network. This includes traffic entering and leaving the virtual environment as well as traffic passed through the WireGuard VPN. Fig.5.1.1 displays a few capabilities of the software, including presentation of all captured packets, filtering packets by content and recognition of the ECH extension in a ClientHello message.



```
not udp
```

```
N Time          Source        Destination   Protocol   Length Info
  0.012089999   172.0.0.1     172.0.0.254   TLSv1.2       225 Application Data
  0.012258929   172.0.0.254   172.0.0.1     TCP            66 443 → 55654 [ACK] Seq=670 Ack=1291 Win=251 Len=0 TSval=4193566044 TSecr=2274294916
  0.012661668   172.0.0.254   172.0.0.1     TLSv1.2       211 Application Data
  0.012806825   172.0.0.254   172.0.0.1     TLSv1.2       182 Application Data
  0.012840910   172.0.0.1     172.0.0.254   TCP            66 55654 → 443 [ACK] Seq=1291 Ack=931 Win=501 Len=0 TSval=2274294917 TSecr=4193566044
  0.013230355   172.0.0.1     172.0.0.2     TCP            74 45712 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=296673615 TSecr=0 WS=128
  0.154706391   172.0.0.2     172.0.0.1     TCP            74 443 → 45712 [SYN, ACK] Seq=0 Ack=1 Win=31856 Len=0 MSS=1460 SACK_PERM TSval=4156540317 TSecr=296673615 WS=128
  0.154727199   172.0.0.1     172.0.0.2     TCP            66 45712 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=296673757 TSecr=4156540317
  0.155504851   172.0.0.1     172.0.0.2     TLSv1.3       665 Client Hello (SNI=dcu.example.com)
  0.292989570   172.0.0.2     172.0.0.1     TCP            66 443 → 45712 [ACK] Seq=1 Ack=600 Win=31872 Len=0 TSval=4156540459 TSecr=296673757
  0.899932151   172.0.0.2     172.0.0.1     TLSv1.3       941 Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data
  0.899974530   172.0.0.1     172.0.0.2     TCP            66 45712 → 443 [ACK] Seq=600 Ack=876 Win=64128 Len=0 TSval=296674502 TSecr=4156541072
  0.924457756   172.0.0.1     172.0.0.2     TLSv1.3       130 Change Cipher Spec, Application Data
  0.925636819   172.0.0.1     172.0.0.2     TLSv1.3       236 Application Data
```

```
› Internet Protocol Version 4, Src: 172.0.0.1, Dst: 172.0.0.2
› Transmission Control Protocol, Src Port: 45712, Dst Port: 443, Seq: 1, Ack: 1, Len: 599
⌄ Transport Layer Security
  ⌄ TLSv1.3 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 594
    ⌄ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 590
        Version: TLS 1.2 (0x0303)
        Random: 81cd16282948e3ab2629c874dd6183270712517d5a29034175f8a43a752f71d8
        Session ID Length: 32
        Session ID: dba8bee60f573ae50b34fc704dae31585f71c23f99c2c5067b731751c6f5663e
        Cipher Suites Length: 34
      › Cipher Suites (17 suites)
        Compression Methods Length: 1
      › Compression Methods (1 method)
        Extensions Length: 483
      › Extension: server_name (len=20) name=dcu.example.com
      › Extension: extended_master_secret (len=0)
      › Extension: renegotiation_info (len=1)
      › Extension: supported_groups (len=14)
      › Extension: ec_point_formats (len=2)
      › Extension: session_ticket (len=0)
      › Extension: application_layer_protocol_negotiation (len=14)
      › Extension: status_request (len=5)
      › Extension: delegated_credentials (len=10)
      › Extension: key_share (len=107) x25519, secp256r1
      › Extension: supported_versions (len=5) TLS 1.3, TLS 1.2
      › Extension: signature_algorithms (len=24)
      › Extension: psk_key_exchange_modes (len=2)
      › Extension: record_size_limit (len=2)
      ⌄ Extension: encrypted_client_hello (len=217)
          Type: encrypted_client_hello (65037)
          Length: 217
          Client Hello type: Outer Client Hello (0)
        › Cipher Suite: HKDF-SHA256/AES-128-GCM
          Config Id: 7
          Enc length: 32
          Enc: 9f8fea85222510bb8cb2f14017fb7ba18e595992be9d25ff33872abe0e826839
```

Figure 5.1.1: Screenshot of Wireshark after capturing a Client Hello message using ECH.

5.2 Evaluation

In this section we use the data captured using WireShark to consider and criticise various parts of the system in operation. The collected data PCAP files have been included in the additional data ZIP archive submitted alongside this report.

5.2.1 Load Distribution

The design appears to have achieved its objective of delivering load fairly across members of the co-operative network. Fig. 5.2.1 shows a very stable load persisting for 25 minutes. In this scenario, 60% of traffic was destined to the blue origin server, while the other two servers shared the remaining 40%. Due to the functionality of the dynamic DNS service, this difference in popularity is not noticeable. While this implementation is extremely simple and does not attempt to make intelligent decisions based on live traffic flow recordings, it does provide preliminary evidence that a dynamic DNS service can be used as a distribution mechanism for the distributed deployment of ECH.

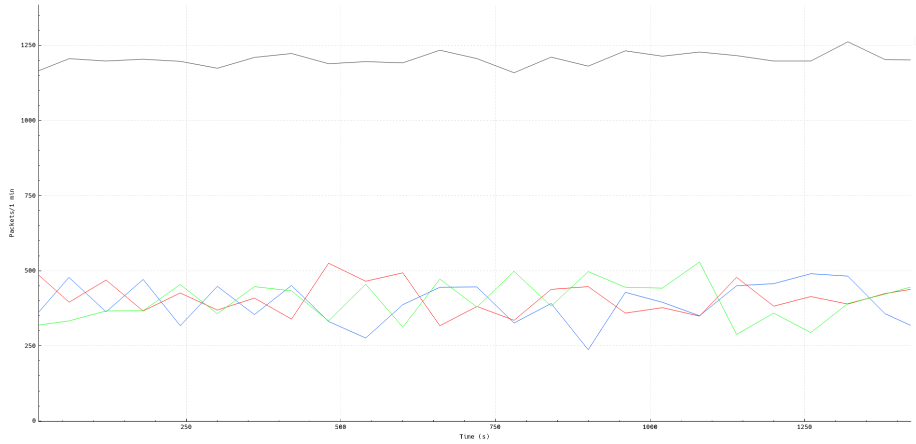


Figure 5.2.1: Load experienced by three co-operative TLS servers from clients over a period of 25 minutes. The black line represents the total load bared across the whole network.

5.2.2 Performance

Data was also collected regarding the performance of scenarios where clients ignore ECH support and go straight to the origin and conventional ECH using a single centralised provider. We can see in Fig. 5.2.2 and Table 5.1 that the distributed deployment model faired just as well as the centralised deployment model. When compared against no ECH usage, clearly the addition of traffic padding and a intermediate node significantly impacts performance. However, this is to be expected as ECH necessitates a intermediate node when using Split Mode. Overall, this result is a great success for the deployment model.

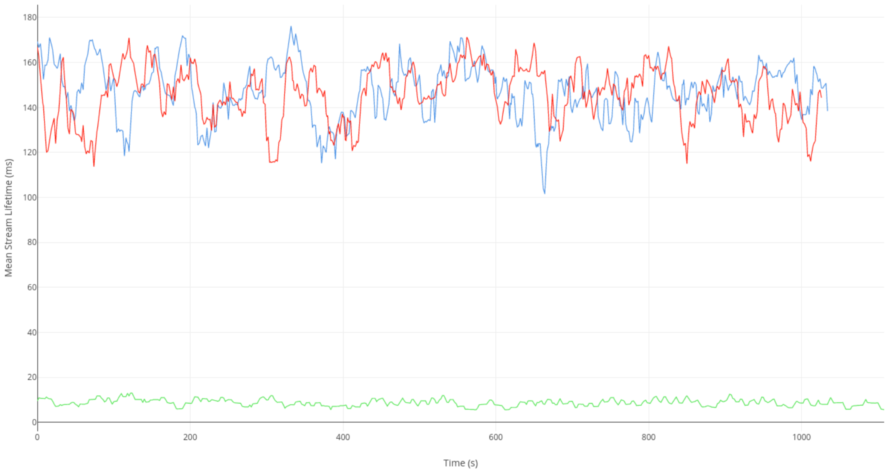


Figure 5.2.2: Total time taken to fetch a resource within different scenarios. Blue is using distributed ECH, red is using centralised ECH and green is disregarding ECH and going straight to the origin server.

Statistic	Distributed ECH	Centralised ECH	Without ECH
# streams (packets)	1054 (11092)	1046 (11017)	1128 (11864)
Minimum lifetime	13.7682 ms	40.6940 ms	4.6399 ms
Maximum lifetime	255.2350 ms	254.2059 ms	23.5630 ms
Mean lifetime	147.4227 ms	146.3109 ms	8.8034 ms
Standard deviation	40.6287 ms	38.6486 ms	5.1609s ms

Table 5.1: Table of performance characteristics

5.2.3 Security

As mentioned previously, the traffic obfuscation aspect of this project was not able to finalise an implementation in time. However, we can still see its attempt at preventing an obvious correlation between the red and blue server transmissions in Fig. 5.2.3. Unfortunately, the current implementation of traffic obfuscation does not mesh well with WireGuard, as a key exchange occurs before tcpdump has time to notice channel activity. Thus, there is always a clear relationship between the two communicators. Another method not implemented but researched is the constant injection of noise that adapts in volume with activity on the channel.

5.3 Summary

The results from this study appear to be promising. The mechanism used for load balancing has archived its objective and distributed deployment does not appear to be negatively effecting network performance. While WireGuard has successfully concealed the ClientHelloInner when being forwarded, it still remains to be seen if the protocol can be operated in such a scenario without being compromised through traffic correlation.

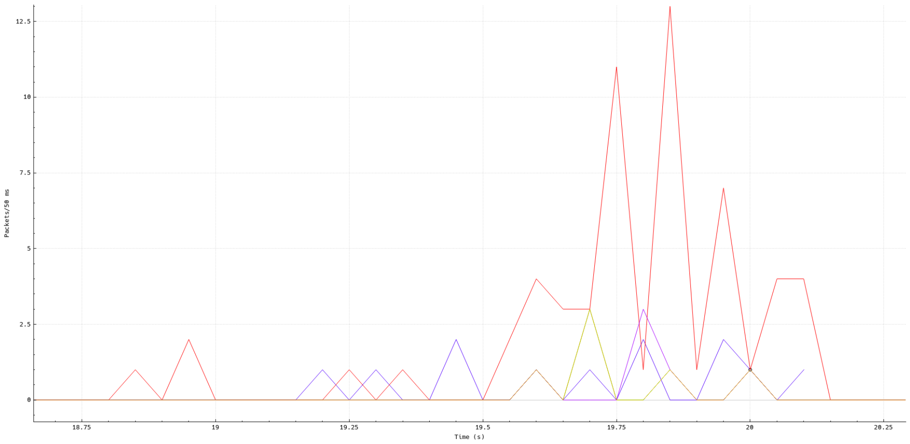


Figure 5.2.3: Impact of traffic obfuscation

6 Conclusion

This research has been a first effort on defining a distributed deployment model for ECH. We have covered an overview of the background technology and concepts needed to appreciate the scope of the work. A study of the design of the deployment model with respect to the challenges surmounted was presented, which was followed by a dive into how the design can be implemented within a virtual testing environment. Lastly, an analysis of the results produced by the testing environment is provided, in which the quality of the design and implementation is evaluated and discussed.

In this final chapter, a summary of what was learnt from this project is included, as well as where this research could benefit from future work. I complete with a short reflection on the project as a whole.

6.1 Learnings

This paper confirms that ECH using Split Mode topology allows for a distributed deployment amongst co-operating TLS servers. We saw that ECH-service load can be distributed evenly across servers using a static DNS configuration with a shared ECHConfig or balanced fairly when using a dynamic DNS service with separate ECHConfigs. There was minimal performance impact observed when compared to a centralised ECH deployment model, but higher latencies and bandwidth usage should be expected when using stricter traffic pacing and mixing parameters. While normalising of co-operating server traffic would ensure perfect masking of client activity, it is generally impractical to achieve this in civilian settings. However, there is evidence that traffic pacing and mixing exhibits sufficient anonymity properties but may be susceptible to statistical pattern detection using a well-trained machine learning model.

6.2 Future Work

The research completed on the security properties of pacing and mixing co-operating server traffic to disrupt correlation attacks is not considered conclusive and requires a follow-up study. It is likely information theory could be employed here to help identify an optimal obfuscation method that minimises traffic impedance and bandwidth usage for a given set of channel throughputs.

Additionally, further work is necessary to determine a shared DNS publication strategy which permits each server to perform regular ECH key rotation. The current design lacks any mechanism for co-operating servers to be able to publish a new set of resource records. In a similar vein, it would be beneficial for the dynamic DNS service to be notified of traffic flow experienced by each server for fairer load balancing. It seems likely both of these tasks would be suitable to be addressed in the same body of work.

In any case, future development of this deployment model should be subjected to more realistic testing environments. This would be expected to include traffic flows and network topologies representative of real-world scenarios. This project has only demonstrated deployment using a single suite of software, namely NGINX and BIND within a QEMU virtual environment, so it may also be reasonable to trial its operation across different configurations of software and hardware.

6.3 Reflection

I am grateful to have been granted considerable freedom within the scope of the project to investigate a number of relevant fields and technologies. I had not previously had the opportunity to work with QEMU or DebVM, so I am very pleased with the reproducible virtual build and testing environments I was able to orchestrate.

However, I must also acknowledge that in covering this additional material within the limited time available, I feel I was only able to explore some areas superficially. Had I more time, I would have liked to continue my research into traffic analysis, correlation attacks and practical countermeasures.

Overall, I found this project to be enjoyable to work on and served as a compelling dissertation topic.

Bibliography

[1] Eric Rescorla et al. *TLS Encrypted Client Hello*. Internet-Draft draft-ietf-tls-esni-18. Work in Progress. Internet Engineering Task Force, Mar. 2024. 51 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-tls-esni/18/.

[2] Zisis Tsiatsikas, Georgios Karopoulos, and Georgios Kambourakis. "Measuring the adoption of TLS encrypted client hello extension and its forebear in the wild". In: *European Symposium on Research in Computer Security*. Springer. 2022, pp. 177–190. DOI: 10.1007/978-3-031-25460-4_10.

[3] Christopher Wood Achiel van der Mandele Alessandro Ghedini and Rushil Mehra. *Encrypted Client Hello - the last puzzle piece to privacy*. Sept. 2023. URL: https://blog.cloudflare.com/announcing-encrypted-client-hello (visited on 03/24/2024).

[4] Mark Nottingham. *Centralization, Decentralization, and Internet Standards*. RFC 9518. Dec. 2023. DOI: 10.17487/RFC9518. URL: https://www.rfc-editor.org/info/rfc9518.

[5] Chia-ling Chan et al. "Monitoring TLS adoption using backbone and edge traffic". In: *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2018, pp. 208–213. DOI: 10.1109/INFCOMW.2018.8406957.

[6] Let's Encrypt Stats. *Percentage of Web Pages Loaded by Firefox Using HTTPS*. URL: https://letsencrypt.org/stats/#percent-pageloads (visited on 04/01/2024).

[7] Christopher Allen and Tim Dierks. *The TLS Protocol Version 1.0*. RFC 2246. Jan. 1999. DOI: 10.17487/RFC2246. URL: https://www.rfc-editor.org/info/rfc2246.

[8] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018. DOI: 10.17487/RFC8446. URL: https://www.rfc-editor.org/info/rfc8446.

[9] Kathleen Moriarty and Stephen Farrell. *Deprecating TLS 1.0 and TLS 1.1*. RFC 8996. Mar. 2021. DOI: 10.17487/RFC8996. URL: https://www.rfc-editor.org/info/rfc8996.

[10] Ralph Holz et al. "The era of TLS 1.3: Measuring deployment and use with active and passive methods". In: *ACM SIGCOMM Computer Communication Review* 50 (3 Aug. 2019), pp. 3–15. DOI: 10.48550/arXiv.1907.12762.

[11] Hyunwoo Lee, Doowon Kim, and Yonghwi Kwon. "TLS 1.3 in practice: How TLS 1.3 contributes to the internet". In: *Proceedings of the Web Conference 2021*. 2021, pp. 70–79. DOI: 10.1145/3442381.3450057.

[12] Peter Hesse et al. *Internet X.509 Public Key Infrastructure: Certification Path Building*. RFC 4158. Sept. 2005. DOI: 10.17487/RFC4158. URL: https://www.rfc-editor.org/info/rfc4158.

[13] Richard Barnes et al. *Automatic Certificate Management Environment (ACME)*. RFC 8555. Mar. 2019. DOI: 10.17487/RFC8555. URL: https://www.rfc-editor.org/info/rfc8555.

[14] Paul Mockapetris. *Domain names - concepts and facilities*. RFC 1034. Nov. 1987. DOI: 10.17487/RFC1034. URL: https://www.rfc-editor.org/info/rfc1034.

[15] Paul Mockapetris. *Domain names - implementation and specification*. RFC 1035. Nov. 1987. DOI: 10.17487/RFC1035. URL: https://www.rfc-editor.org/info/rfc1035.

[16] Stéphane Bortzmeyer. *DNS Privacy Considerations*. RFC 7626. Aug. 2015. DOI: 10.17487/RFC7626. URL: https://www.rfc-editor.org/info/rfc7626.

[17] Alissa Cooper et al. *Privacy Considerations for Internet Protocols*. RFC 6973. July 2013. DOI: 10.17487/RFC6973. URL: https://www.rfc-editor.org/info/rfc6973.

[18] Stephen Farrell and Hannes Tschofenig. *Pervasive Monitoring Is an Attack*. RFC 7258. May 2014. DOI: 10.17487/RFC7258. URL: https://www.rfc-editor.org/info/rfc7258.

[19] Zi Hu et al. *Specification for DNS over Transport Layer Security (TLS)*. RFC 7858. May 2016. DOI: 10.17487/RFC7858. URL: https://www.rfc-editor.org/info/rfc7858.

[20] Paul E. Hoffman and Patrick McManus. *DNS Queries over HTTPS (DoH)*. RFC 8484. Oct. 2018. DOI: 10.17487/RFC8484. URL: https://www.rfc-editor.org/info/rfc8484.

[21] Sebastián García et al. "Large scale measurement on the adoption of encrypted DNS". In: *arXiv e-prints* (July 2021). DOI: 10.48550/arXiv.2107.04436.

[22] Benjamin M. Schwartz, Mike Bishop, and Erik Nygren. *Service Binding and Parameter Specification via the DNS (SVCB and HTTPS Resource Records)*. RFC 9460. Nov. 2023. DOI: 10.17487/RFC9460. URL: https://www.rfc-editor.org/info/rfc9460.

[23] Richard Barnes et al. *Hybrid Public Key Encryption*. RFC 9180. Feb. 2022. DOI: 10.17487/RFC9180. URL: https://www.rfc-editor.org/info/rfc9180.

[24] Karthikeyan Bhargavan, Vincent Cheval, and Christopher Wood. "A symbolic analysis of privacy for TLS 1.3 with Encrypted Client Hello". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2022, pp. 365–379. DOI: 10.1145/3548606.3559360.

[25] Adam Back, Ulf Möller, and Anton Stiglic. "Traffic analysis attacks and trade-offs in anonymity providing systems". In: *International Workshop on Information Hiding*. Springer. 2001, pp. 245–257. DOI: 10.1007/3-540-45496-9_18.

[26] Samuel Padraic DeFabbia-Kane. "Analyzing the effectiveness of passive correlation attacks on the tor anonymity network". In: (2011). DOI: 10.14418/wes01.1.1636.

[27] Martino Trevisan et al. “Attacking DoH and ECH: Does Server Name Encryption Protect Users’ Privacy?” In: *ACM Transactions on Internet Technology* 23.1 (2023), pp. 1–22. DOI: 10.1145/3570726.

[28] Brian N Levine et al. “Timing attacks in low-latency mix systems”. In: *Financial Cryptography: 8th International Conference, FC 2004, Key West, FL, USA, February 9-12, 2004. Revised Papers 8*. Springer. 2004, pp. 251–265. DOI: 10.1007/978-3-540-27809-2_25.

[29] Charles V Wright, Scott E Coull, and Fabian Monrose. “Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis.” In: *NDSS*. Vol. 9. 2009.

[30] Shui Yu et al. “Predicted packet padding for anonymous web browsing against traffic analysis attacks”. In: *IEEE Transactions on Information Forensics and Security* 7.4 (2012), pp. 1381–1393. DOI: 10.1109/TIFS.2012.2197392.

[31] Xinwen Fu et al. “Analytical and empirical analysis of countermeasures to traffic analysis attacks”. In: *2003 International Conference on Parallel Processing, 2003. Proceedings.* IEEE. 2003, pp. 483–492. DOI: 10.1109/ICPP.2003.1240613.

[32] Xinwen Fu et al. “On effectiveness of link padding for statistical traffic analysis attacks”. In: *23rd International Conference on Distributed Computing Systems, 2003. Proceedings.* IEEE. 2003, pp. 340–347. DOI: 10.1109/ICDCS.2003.1203483.

[33] Fabrice Bellard. “QEMU, a fast and portable dynamic translator.” In: *USENIX annual technical conference, FREENIX Track*. Vol. 41. 46. 2005.

[34] Shelena Soosay Nathan et al. “BERKELEY INTERNET NAME DOMAIN (BIND)”. In: *International Journal on Cybernetics & Informatics* 1 (Feb. 2012), pp. 1–10.

[35] Paul A. Vixie et al. *Dynamic Updates in the Domain Name System (DNS UPDATE)*. RFC 2136. Apr. 1997. DOI: 10.17487/RFC2136. URL: https://www.rfc-editor.org/info/rfc2136.

[36] Jason A Donenfeld. “WireGuard: Next Generation Kernel Network Tunnel.” In: *NDSS*. 2017, pp. 1–12. DOI: 10.14722/ndss.2017.23160.

[37] Werner Almesberger. *Linux Network Traffic Control - Implementation Overview*. Apr. 1999.

[38] Stephen Hemminger. “Network emulation with NetEm”. In: *Linux conf au*. Vol. 5. Apr. 2005.

A1 Project Files

For the sake of experimental reproducibility, a ZIP archive of the project code, three scenarios and packet capture files have been provided alongside this report. Additionally, a copy can be found online: https://github.com/tedski999/distributed-ech.

The source code consists of a single Bash script, `run.sh`, which contains all the necessary logic to setup and run any virtual testing environment described using two scenario configuration files, `network.csv` and `server.csv`. The script is dependent on DebVM, and by extension QEMU and mmdebstrap. A path `sandbox` must also be specified as the directory to store QEMU images and other ephemeral data.

Given two scenario configuration files, the environment can be generated and started by executing `./run.sh sandbox network.csv servers.csv`. Initial setup times can be lengthy, as OpenSSL, curl and NGINX must be built before all QEMU virtual machine images are configured and booted. These builds and configurations are preserved, so later environment boot up times are far quicker.

Once running, a virtual machine can be accessed using the corresponding SSH command printed to the terminal: `ssh -i 'sandbox/ssh.key' -p 2222 root@127.0.0.1`. See Chapter 4 for more information on how to use this environment.

A2 Verbose curl Output

```
root@tls-client:~# curl --verbose --cacert /keys/root.crt --ech hard --doh-url https://dns.example.com/dns-query https://tcd.example.com
* Some HTTPS RR to process
* Host tcd.example.com:443 was resolved.
* IPv6: (none)
* IPv4: 172.0.0.2
*   Trying 172.0.0.2:443...
* Connected to tcd.example.com (172.0.0.2) port 443
* ECH: ECHConfig from DoH HTTPS RR
* ECH: imported ECHConfigList of length 68
* ALPN: curl offers h2,http/1.1
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
*  CAfile: /keys/root.crt
*  CApath: /etc/ssl/certs
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384 / x25519 / id-ecPublicKey
* ECH: result: status is succeeded, inner is tcd.example.com, outer is dcu.example.com
* ALPN: server accepted h2
* Server certificate:
*  subject: CN=tcd.example.com
*  start date: Apr 10 18:05:25 2024 GMT
*  expire date: Apr  8 18:05:25 2034 GMT
*  subjectAltName: host "tcd.example.com" matched cert's "tcd.example.com"
*  issuer: CN=root.example.com
*  SSL certificate verify ok.
*   Certificate level 0: Public key type EC/secp384r1 (384/192 Bits/secBits), signed using ecdsa-with-SHA256
*   Certificate level 1: Public key type EC/secp384r1 (384/192 Bits/secBits), signed using ecdsa-with-SHA256
* using HTTP/2
* [HTTP/2] [1] OPENED stream for https://tcd.example.com/
* [HTTP/2] [1] [:method: GET]
* [HTTP/2] [1] [:scheme: https]
* [HTTP/2] [1] [:authority: tcd.example.com]
* [HTTP/2] [1] [:path: /]
* [HTTP/2] [1] [user-agent: curl/8.7.2-DEV]
* [HTTP/2] [1] [accept: */*]
> GET / HTTP/2
> Host: tcd.example.com
> User-Agent: curl/8.7.2-DEV
> Accept: */*
>
* Request completely sent off
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* old SSL session ID is stale, removing
< HTTP/2 200
< server: nginx/1.25.4
< date: Wed, 10 Apr 2024 19:12:44 GMT
< content-type: text/html
<
<!doctype html>
<html lang=en>
  <head>
    <meta charset=utf-8>
    <title>tcd.example.com</title>
  </head>
  <body>
    <img src="/image.png" width="300" height="300">
    <p>
      Welcome to <b>tcd.example.com</b><br/>
      Got here via <i>172.0.1.2</i>
    </p>
    <ul>
      <li>SNI: tcd.example.com</li>
      <li>HTTP host: tcd.example.com</li>
      <li>ALPN protocol: h2</li>
    </ul>
    <form action="/pkglist">
      <input type="submit" value="Download pkglist" />
    </form>
    <p>
      Sites on dcu (172.0.0.2):<br/>
      <a href="https://dcu.example.com">dcu.example.com</a>
    </p>
    <p>
      Sites on tcd (172.0.0.5):<br/>
      <a href="https://tcd.example.com">tcd.example.com</a> *
    </p>
    <p>
      Sites on ucd (172.0.0.8):<br/>
      <a href="https://ucd.example.com">ucd.example.com</a>
    </p>
  </body>
</html>
* Connection #0 to host tcd.example.com left intact
```