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 <seeks> to plug a few remaining privacy leaks , but <>

Acknowledgements

This work has been <>

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.

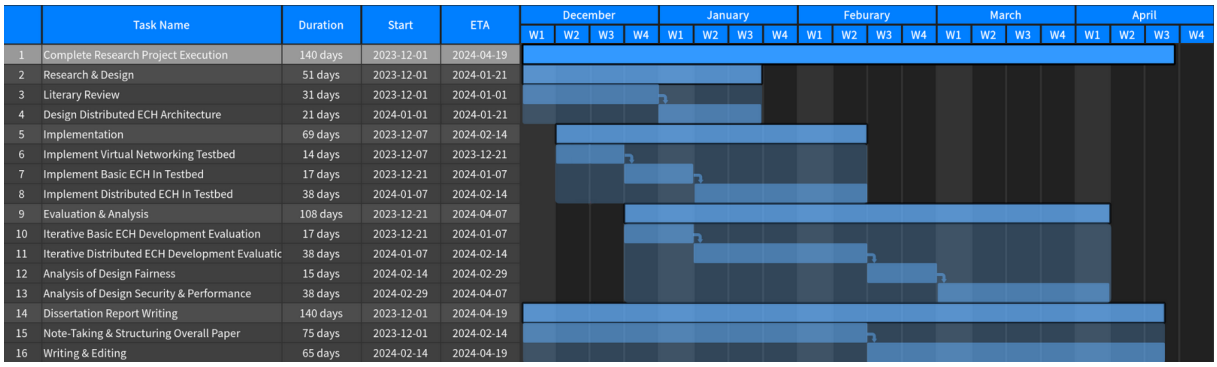
2. **Design, evalute and contrast deployment models.** An iterative development process is used to produce a series of incrementally improved skeletal prototypes, with the goal to rapidly design and test for functionality guided by the design criteria and results of previous work as heuristics.

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.

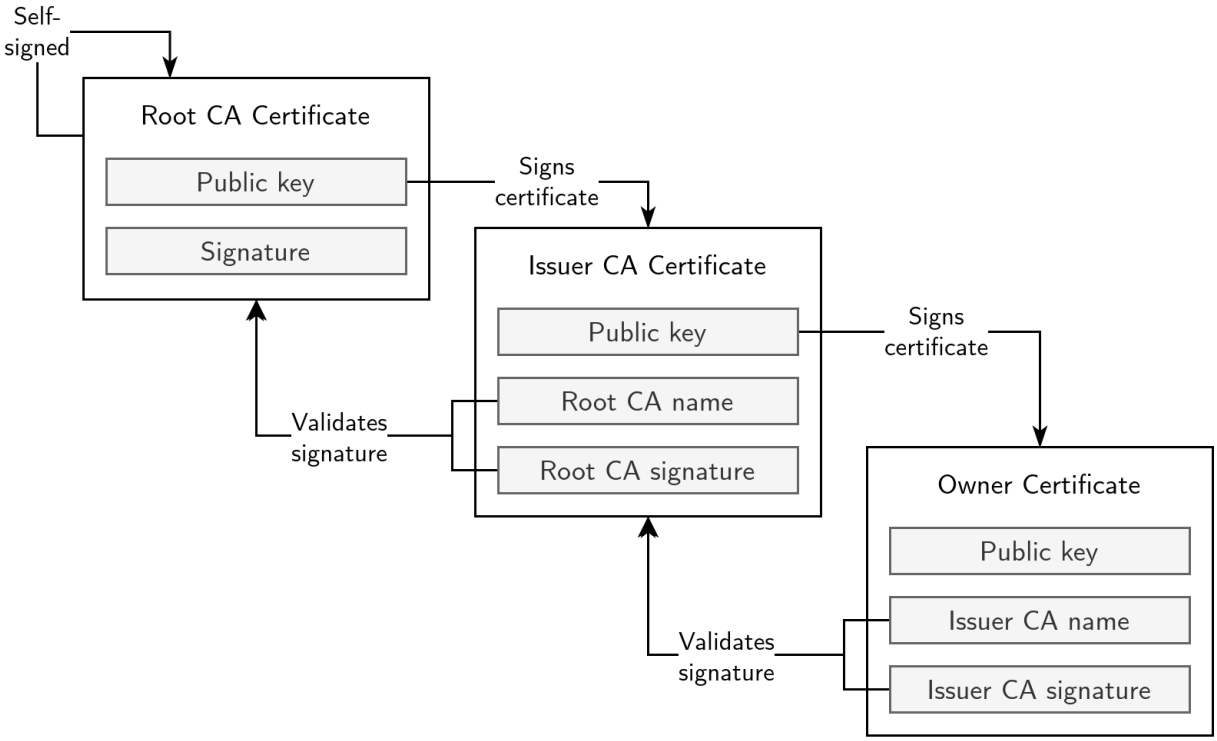


Figure 2.1.1: A chain of trust established between the unknown owner certificate and the implicitly trusted root CA certificate in order to authenticate the identity of the owner against its associated public key.

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 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.

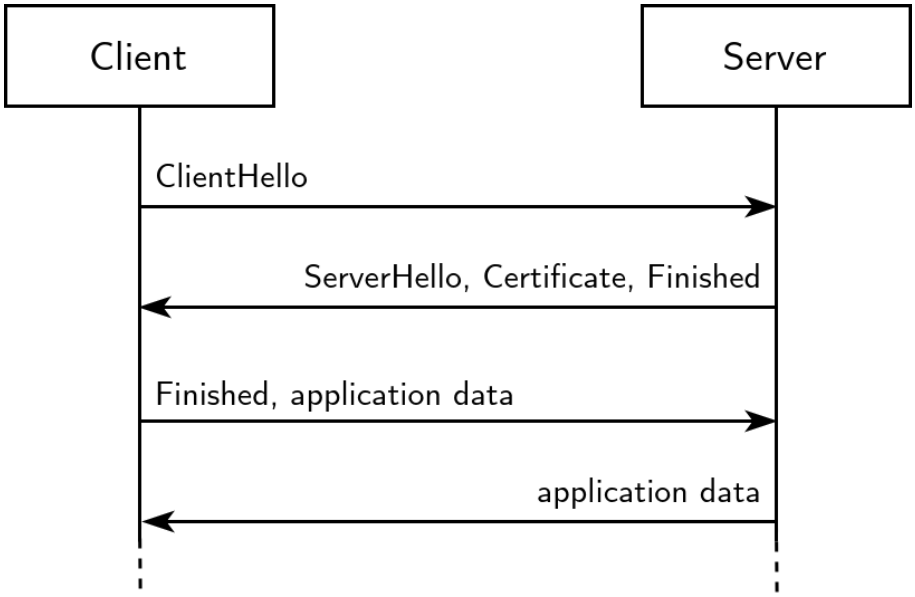


Figure 2.1.2: Sequence diagram between a client and server describing a basic TLS 1.3 handshake with only server authentication.

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 and several extensions are 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.

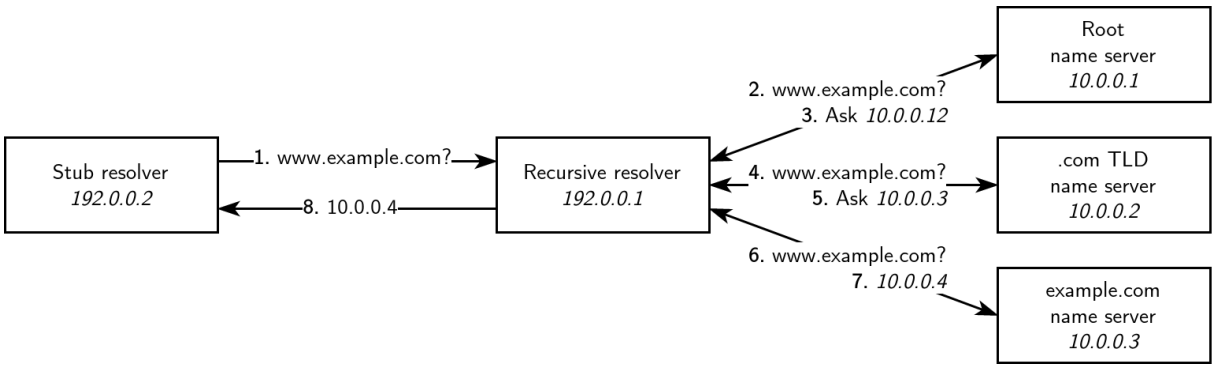


Figure 2.2.1: A stub resolver requests a recursive resolver to retrieve resource records for 'www.example.com'. Without previously cached query responses, the recursive resolver must navigate the domain name hierarchy starting at a known root name server.

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 recommends 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 lead 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.

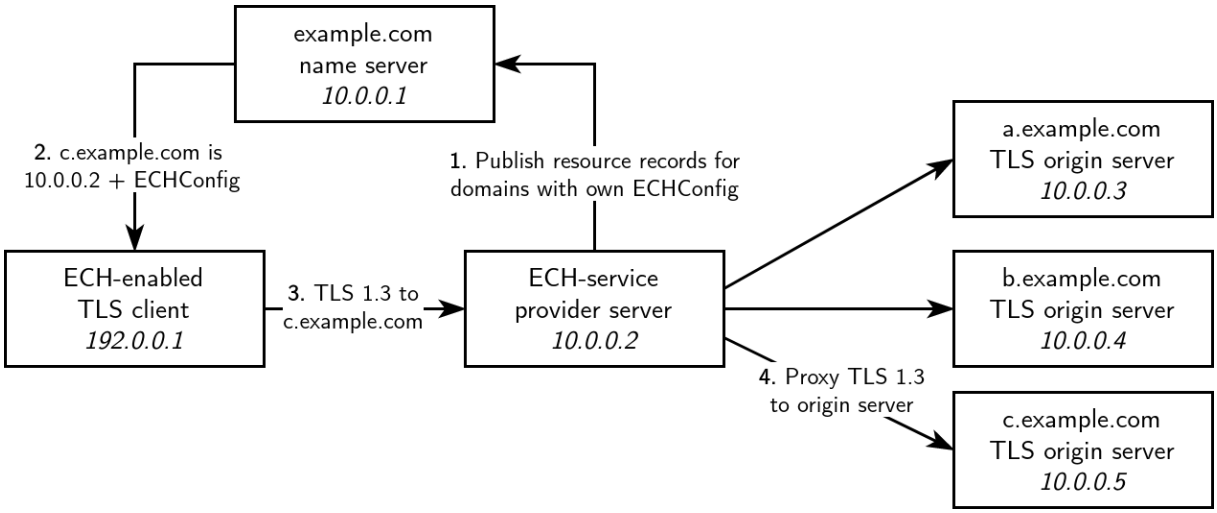


Figure 2.3.1: An example of how ECH can be used in Split Mode when the ECH-service provider is not co-located with the requested private domain origin server. Steps 1 and 2 are completed out-of-band. Steps 3 and 4 result in a TLS connection being established between the client and origin server.

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 disseration aims to serve as a guide for security researchers and service operator on the viablity of distributed ech> <during the course of this project, a system design was formulated and iteratively refined> <the main parts of the design considered are its deployment schema and protecting traffic>

<in this chapter, we will study the determined solution as well as the challenges that motivated its design> <this consists of a overview of purposed system followed by an examination of its individual components>

3.1 Problem Overview

<ech split mode serves as a bases for distributed deployment, but makes no attempt to address the implications of servers being operated by separate organisations and located across a public network> <to do so, we must consider a number of challenges faced by this situation to enable co-operation and secure functioning> <this paper purposes a loose network of tls servers all acting as ech providers for each other which proxy connections to the true origin server, as depicted in Fig. 3.1.1>

<parallel to the operation of ech split mode> <loose network must publish appropriate resource records such that clients choose to query any of them for any of their domains> <all providers listens on their public facing network interface for with ClientHelloOuter> <providers also listen for actual connections> <clients resolve requested domain using DoH and then use round robin or other selection process to uniformly choose one of the providers at random with the correct echconfig> <client begins establishment to the requested via the selected provider using the providers echconfig> <as in ech split mode, the provider decrypts the ClientHelloInner to retrieve the actual domain requested> <the provider maps the domain to the true origin server and proxies the clients connection over communication network to the origin server> <the origin server completes the tls handshake with the client through the provider> <this connection is end-to-end encrypted, so the provider does not mitm>

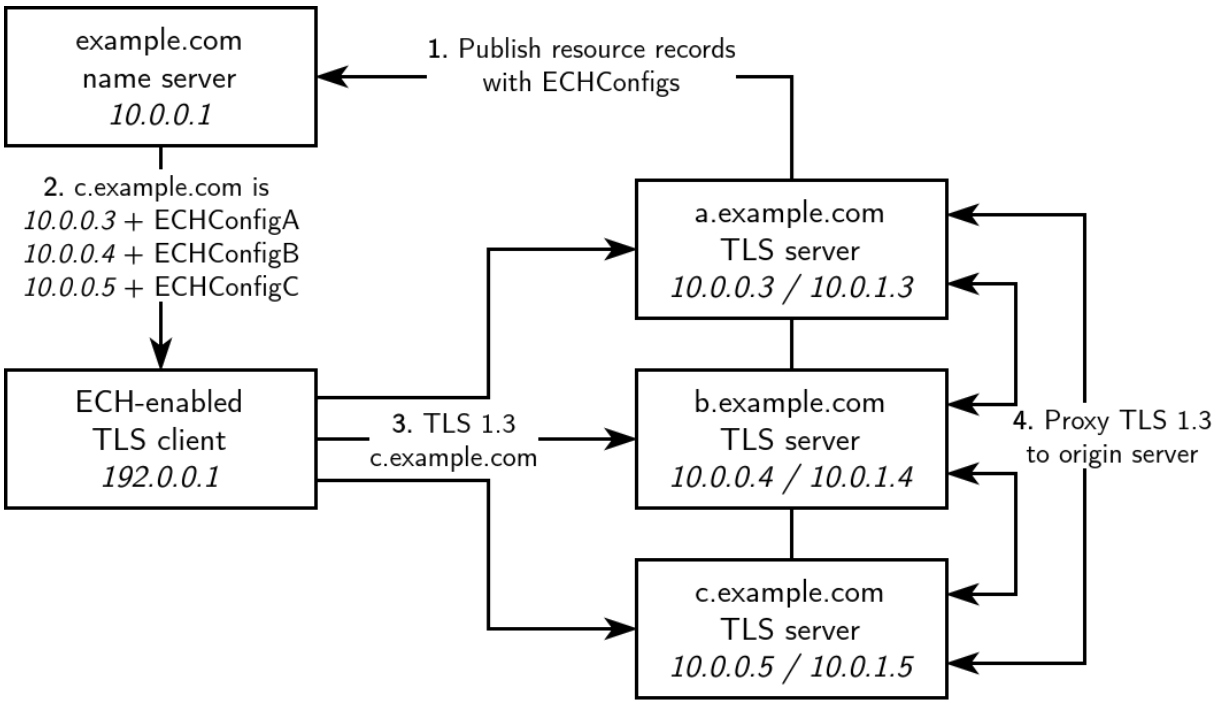


Figure 3.1.1: <>

<the design challenges can be split into two topics, mechanism for distribution and traffic obfuscation> <the distribution mechanism deals with a couple designs for the a dns publication schema so compatible clients follow the above process, and the networking required for tls servers to communicate with each other>

3.2 Distribution Mechanism

<distribution is based on two parts> <first, clients need to be able to select from several options provided by a dns query> <second, co-oerating tcp servers need to be able to forward connections to each other based on the decrypted ClientHelloInner>

3.2.1 DNS Publication Schema

<one approach: shared echconfig with round robin A resource records> <load is evenly distributed across servers because client uses round robin on selected A rrs> <but requires shared secrets between all servers as there is no way to specify which ech key is associated with which host>

<a better approach: using alternative endpoints to associate ech keys with individual servers> <load is evenly distributed across servers due to matching priority> <unfortunately more fine-grained load distribution is not possible without srv resource records adoption> <we can replicate this using a dynamic dns service, where records are regularly substituted such that load is distributed across servers in a fair manner>

3.2.2 TLS Server Co-operation

<tls servers are physically separate from each other and must communicate over the same network as the client when forwarding the client connection> <servers still listen for normal tls connections on 443 port of the public interface as well as ech-enabled connection> <ech split mode connections have their ClientHelloInner decrypted and private domain mapped to the actual origin> <clients connection is forwarded to the origin server> <the origin server listening on the vpn interface completes the tls handshake with the client through the provider>

3.3 Traffic Obfuscation

<ech is susceptible, classifiers>[27] <correlation attacks in low-latency systems like web browsing and video streaming compared to high-latency like email>[28] <rx/tx timing> <packet lengths> <traffic patterns> [26]

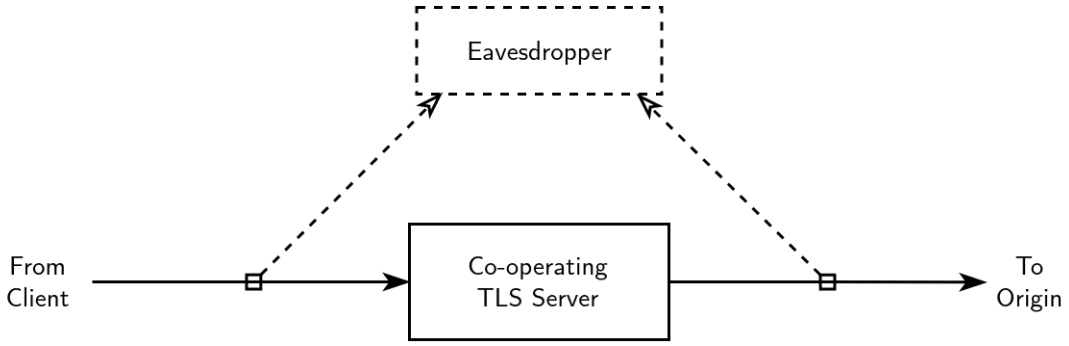


Figure 3.3.1: <TODO>

<traffic obfuscation techniques lead "to disrupt the patterns"> <encrypt traffic between participating servers on top of the public network>

3.3.1 Normalisation

<shannon and perfect secrecy: removal of all identifying features> <chatter> <injection of dummy traffic to fill bandwidth gaps> <impracticality in civilian environments due to required bandwidth>

3.3.2 Pacing and Mixing

<while not perfect, many practical techniques exist to mask traffic with minimal impacts> <padding packets themselves to prevent packet length correlation> [30] <to disrupt timing-based correlation attacks, delays and slotting (aka pacing) can be used> <to disrupt deep packet inspection, other packet-based correlation attacks (traffic patterns), traffic

mixing can be used> [31, 32] <a simple technique is packet duplication, where a similar packet is sent to every peer>

3.4 Summary

<it fulfills the desired design criteria> <we use dns https rrs to distributed load across multiple ech providers> <separate echconfigs can be used be each server as the ech key is associated with the server> <an alternative strategy exists that is more robust but requires shared echconfig> <ech split mode is vulnerable to correlation attacks> <a virtual private network between co-operating servers is used to allow peer communication> <but techniques exist to defend against attacks>

4 Implementation

<this chapter outlines steps taken to simulate a working prototype of the design> <this is used for evaluating the practical challenges faced by design> <project work can be found online, as well as in appendix A>[1]

<look at how we simulate a deterministic, reproducible, configurable, measurable network environment with our own SSL certificates> <how dns server can be configured to provide DoH with the designed publication strategy> <tls server as well as configuration of standard ECH-enabled clients and web browsers>

4.1 Simulation

<simulation overview: qemu+bridge>

4.1.1 Virtualisation

<introduce qemu+debvm> [33] <why problem: build is slow> <build process, separating builder from hosts> <using ssh to automate configuration>

<host base> <boot up process>

4.1.2 Networking

<overview of networking and how it works with qemu> <overview of linux networking devices> <introduction to bridge networks>

<the static network configuration for simulation> <using systemd to setup networking for all virtual machines at boot>

<introduction to openssl for certificates> [**TODO:openssl**] <root certificate generated for simulation>

<server certificates can be issued by the root ca as such>

[1] https://github.com/tedski999/distributed-ech

```
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: TODO builder

```
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: TODO br0

<with this we have a foundation for building networked virtual machines on top of> <the rest of this chapter will be on how we implement these different types of servers and applications>

4.2 DNS Server

<one of the virtual machines operates as the dns server for the client to query> <we use bind9 to provide this service> [**TODO:bind**] <for practical ech client implementations that we will use, they require dns-over-https to use ech> <we enable this using tls certificate as seen in fig>

<bind9 uses zone files to specify> <we will use them to specify all three dns designs> <this first one is the simpler solution based on a records round robin but requiring a shared

```
[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: TODO /etc/systemd/network/00-br0.network contents

```
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: TODO root ca

ech key>

<as mentioned before, this is generally a security no no> <we can do this>

<this is a more involved, might break the spec and has shown mixed results> <a dynamic dns service as such solves both of these issues>

<this script will do this using nsupdate to send DNS zone updates>

<of course, this loses static configuration quality and is more complicated>

4.3 TLS Server

<remaining virtual machines operates as the co-operating tls servers> <these servers are connected together over the bridge network>

<tls certs + ech key>

<use nginx as web server> [34]

<wireguard vpn>[35]

<wireguard configuration>

<wireguard peers>

<use tc to pad traffic> [36, 37]

<all in all, works>

```
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: TODO dns

```
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;
    file "/var/lib/bind/db.example.com";
};
```

Listing 4.2.1: TODO /etc/bind/named.conf contents

4.4 TLS Client

<recent work has been done to add ech support to many common clients> <using several clients for qualitative and quantitative testing> <for this project, we test with curl, firefox, chrome> <we configure these clients to use our dns-over-https and ssl root>

4.4.1 curl

<introduction to curl> <work by farrell> <explanation of commands needed below>

<challenges> <working>

4.4.2 Mozilla Firefox

<introduction to firefox> <explanation of config needed for below>

<challenges> <working>

```
$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

tcd IN A 172.0.0.2
tcd IN A 172.0.0.5
tcd IN A 172.0.0.8

ucd IN A 172.0.0.2
ucd IN A 172.0.0.5
ucd IN A 172.0.0.8

dcu IN HTTPS 1 . ech=<Shared ECHConfig>
tcd IN HTTPS 1 . ech=<Shared ECHConfig>
ucd IN HTTPS 1 . ech=<Shared ECHConfig>
```

Listing 4.2.2

4.4.3 Google Chrome

<introduction to chrome> <explanation of config needed for below>

<challenges> <working>

4.5 Summary

<this chapter has presented an implementation of the design for testing purposes> <simulated using qemu with debvm> <bind9 is used for dns> <nginx is used as web server> <wireguard and tc for traffic obfuscation> <curl, firefox and chrome as clients>

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

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

dcu.ech IN A 172.0.0.2
tcd.ech IN A 172.0.0.5
ucd.ech IN A 172.0.0.8

dcu IN HTTPS 1 dcu.ech ech=<DCU ECHConfig>
dcu IN HTTPS 1 tcd.ech ech=<TCD ECHConfig>
dcu IN HTTPS 1 ucd.ech ech=<UCD ECHConfig>

tcd IN HTTPS 1 dcu.ech ech=<DCU ECHConfig>
tcd IN HTTPS 1 tcd.ech ech=<TCD ECHConfig>
tcd IN HTTPS 1 ucd.ech ech=<UCD ECHConfig>

ucd IN HTTPS 1 dcu.ech ech=<DCU ECHConfig>
ucd IN HTTPS 1 tcd.ech ech=<TCD ECHConfig>
ucd IN HTTPS 1 ucd.ech ech=<UCD ECHConfig>
```

Listing 4.2.3

```
$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
tcd IN A 172.0.0.5
ucd IN A 172.0.0.8

dcu IN HTTPS 1 . ech=<DCU ECHConfig>
tcd IN HTTPS 1 . ech=<TCD ECHConfig>
ucd IN HTTPS 1 . ech=<UCD ECHConfig>
```

Listing 4.2.4

```
# nsupdate
> update delete oldhost.example.com A
> update add newhost.example.com 86400 A 172.16.1.1
> send
```

Listing 4.2.5

```
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: TODO host+site tls

```
stream {
    ssl_preread on;
    ssl_echkeydir /keys/tcd;
    server {
        listen 172.0.0.5:443;
        proxy_pass $backend;
    }
    map $ssl_preread_server_name $backend {
        dcu.example.com 172.0.1.2:443;
        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 index.htm;
        }
    }
}
```

Listing 4.3.2: TODO nginx

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

Listing 4.3.3: TODO host wg

```
[NetDev]
Name=wg0
Kind=wireguard

[WireGuard]
ListenPort=51820
PrivateKey=<TCD WireGuard Private Key>

[WireGuardPeer]
PublicKey=<DCU WireGuard Public Key>
AllowedIPs=172.0.1.2/32
Endpoint=172.0.0.2:51820

[WireGuardPeer]
PublicKey=<UCD WireGuard Public Key>
AllowedIPs=172.0.1.8/32
Endpoint=172.0.0.8:51820
```

Listing 4.3.4: TODO host wg

```
[Match]
Name=wg0

[Network]
Address=172.0.1.5/24
```

Listing 4.3.5: TODO host wg0

```
tc qdisc replace dev enp0s6 root netem slot 100ms 200ms

tcpdump -i wg0 -nnqlt ip and 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 4.3.6

```
LD_LIBRARY_PATH=/mnt/openssl/lib64 /mnt/curl/bin/curl \
    --verbose --cacert /keys/root.crt --ech hard \
    --doh-url https://ns.example.com/dns-query https://tcd.example.com
```

Listing 4.4.1: TODO curl

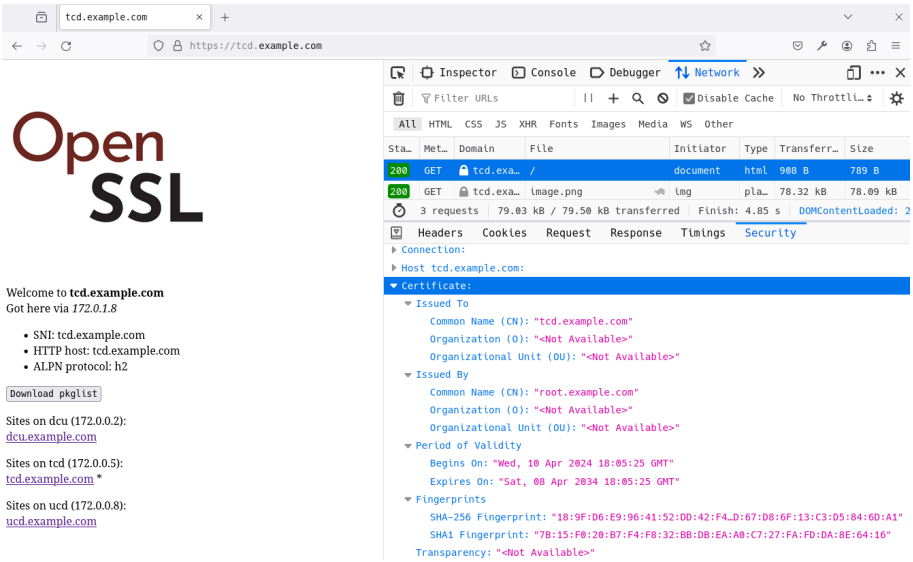


Figure 4.4.1: <TODO>

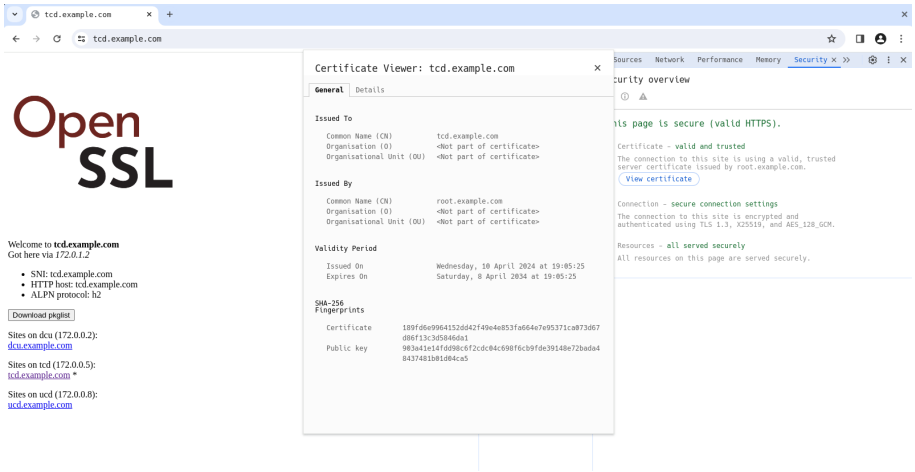


Figure 4.4.2: <TODO>

5 Results and Discussion

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language. <during this project, many

<in the following sections, we will have a look at the

5.1 Data Collection

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

5.2 Evaluation

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

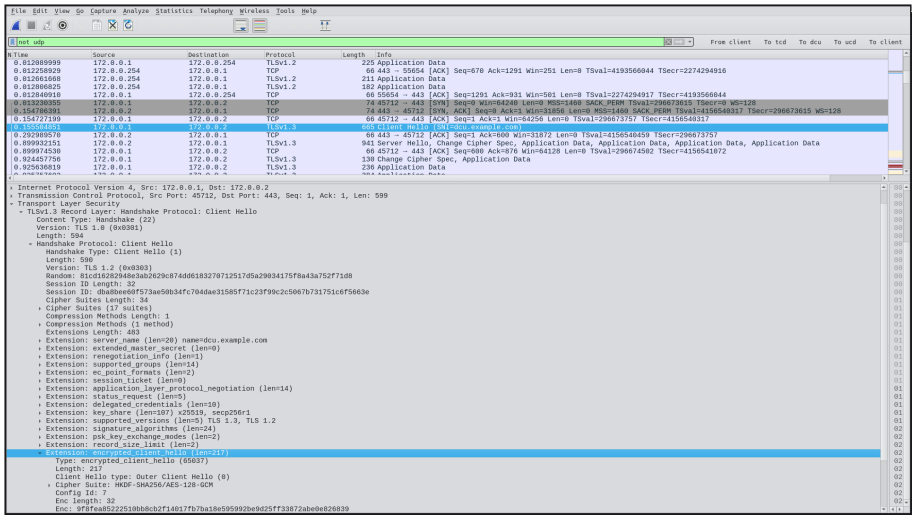


Figure 5.1.1: <>

5.2.1 Performance

compare against normal ech and no ech Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

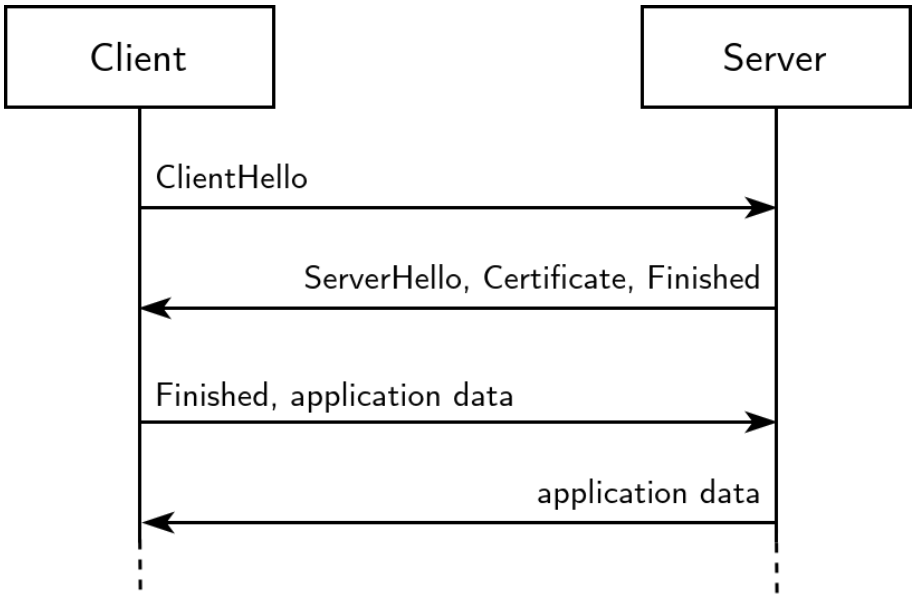


Figure 5.2.1: <latency vs >

5.2.2 Security

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

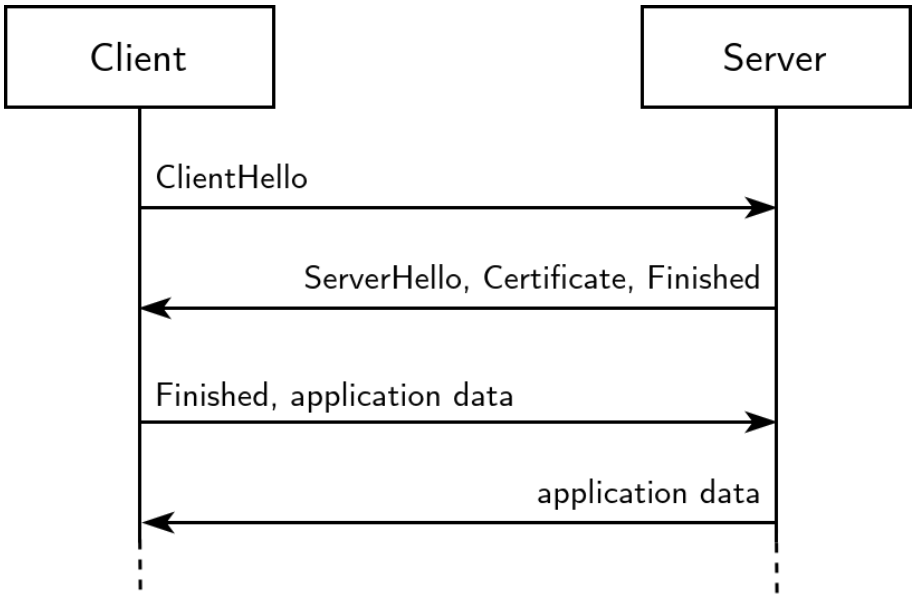


Figure 5.2.2: <>

5.3 Limitations

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

5.3.1 Load Balancing

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest

gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

5.3.2 Traffic Padding

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

5.4 Summary

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

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 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.

[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] Will Reese. "Nginx: the high-performance web server and reverse proxy". In: *Linux Journal* 2008.173 (2008), p. 2.

[35] Jason A Donenfeld. "WireGuard: Next Generation Kernel Network Tunnel." In: *NDSS*. 2017, pp. 1–12.

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

[37] Stephen Hemminger. "Network emulation with NetEm". In: *Linux conf au*. Vol. 5. Apr. 2005.

A1 Project code listing

<what this code is in relation to project> <why it was created> <it does this that and the other thing> <requirements>

Run with `./run.sh sandbox network.csv servers.csv`.

<file purpose>

```
example.com
dns
42:ff:ff:ff:ff:ff
172.0.0.254
```

<file purpose>

```
dcu,42:11:11:11:11:11,172.0.0.2,172.0.1.2,dcu
tcd,42:44:44:44:44:44,172.0.0.5,172.0.1.5,tcd
ucd,42:77:77:77:77:77,172.0.0.8,172.0.1.8,ucd
```

<file purpose>

```
#!/bin/bash

hash sudo ip ssh ssh-keygen debvm-create debvm-run debvm-waitssh || exit 1
msg() { printf "\n\033[1;33m$@\033[0m\n"; }
pkill qemu-system-x86

# Parse provided config
dir="$1"
network_cfg="$(<"$2")" || exit 1
server_cfgs="$(<"$3")" || exit 1
read -d "" domain dns_host dns_mac dns_ip <<< "$network_cfg"
mkdir -p "$dir" || exit 1

# Generate SSH keypair
[ -f "$dir/ssh.key" ] && [ -f "$dir/ssh.key.pub" ] || {
```

```
  msg "Generating ssh keypair for VMs..."
  ssh-keygen -N "" -t ed25519 -f "$dir/ssh.key" || exit 1
}

# Setup host network bridge
sudo ip link show br0 1>/dev/null 2>&1 || {
  msg "Creating network bridge br0..."
  sudo ip link add name br0 type bridge || exit 1
  sudo ip addr add 172.0.0.1/24 dev br0 || exit 1
}
[ -z "$(ip link show br0 up)" ] && {
  msg "Bringing up network bridge br0..."
  sudo ip link set dev br0 up || exit 1
}

# Generate build with builder
[ -f "$dir/builder.img" ] || {
  msg "Generating builder.img..."
  debvm-create -h builder -o "$dir/builder.img" -r unstable -z 2GB -k
  ↪  "$dir/ssh.key.pub" -- \
    --include ca-certificates,build-essential,dh-autoreconf,git,e2fsprogs \
    --include libpsl-dev,libpcre3-dev,libz-dev,libnghttp2-dev || exit 1
}
[ -f "$dir/build.img" ] || {
  msg "Generating build.img:"

  cmds="
  # Format and mount build.img
  mkfs.ext4 -L build /dev/vdb || exit 1
  mount /dev/vdb /mnt || exit 1

  # Build OpenSSL patched with ECH support
  git clone -b ECH-draft-13c https://github.com/sftcd/openssl.git /mnt/src/openssl \
    && cd /mnt/src/openssl || exit 1
  ./config --prefix=/mnt/openssl --openssldir=/mnt/openssl || exit 1
  make -j8 || exit 1
  make -j8 install || exit 1

  # Build curl patched with ECH support
  git clone -b ECH-experimental https://github.com/sftcd/curl.git /mnt/src/curl \
    && cd /mnt/src/curl || exit 1
  autoreconf -fi || exit 1
  CPPFLAGS=-I/mnt/openssl/include LDFLAGS=-L/mnt/openssl/lib64 ./configure \\
    --prefix=/mnt/curl --with-openssl --enable-ech --enable-httpsrr || exit 1
  LD_LIBRARY_PATH=/mnt/openssl/lib64 make -j8 || exit 1
  make -j8 install || exit 1
```

```
61
62    # Build NGINX patched with ECH support
63    git clone -b ECH-experimental https://github.com/sftcd/nginx.git /mnt/src/nginx \
64      && cd /mnt/src/nginx || exit 1
65    ./auto/configure --prefix=/mnt/nginx \\
66      --with-cc-opt=-I/mnt/openssl/include --with-ld-opt=-L/mnt/openssl/lib64 \\
67      --with-stream --with-stream_ssl_module --with-stream_ssl_preread_module \\
68      --with-http_ssl_module --with-http_v2_module || exit 1
69    LD_LIBRARY_PATH=/mnt/openssl/lib64 make -j8 || exit 1
70    make -j8 install || exit 1
71    sed 's/\\/usr\\/sbin\\/nginx/\\/mnt\\/nginx\\/sbin\\/nginx -c \\/site\\/nginx.conf
      ↪  -p \\/site\\/nginx/' \\
72      /mnt/src/nginx/debian/nginx-common.nginx.service > /mnt/nginx/nginx.service ||
        ↪  exit 1
73    >>/mnt/nginx/nginx.service echo '
74    [Service]
75    Environment=LD_LIBRARY_PATH=/mnt/openssl/lib64' || exit 1
76
77    # Graceful shutdown
78    cd / && umount /mnt || exit 1
79    shutdown now"
80
81    echo "$cmds"
82    qemu-img create "$dir/build.img" 2G || exit 1
83    debvm-run --image "$dir/builder.img" --sshport 2222 --graphical -- \
84      -display none -drive file="$dir/build.img",format=raw,if=virtio,readonly=off &
85    debvm-waitssh 2222 || exit 1
86    ssh -o NoHostAuthenticationForLocalhost=yes -i "$dir/ssh.key" -p 2222
      ↪  root@127.0.0.1 "$cmds" || exit 1
87    wait
88  }
89
90  # Generate base VM image
91  [ -f "$dir/base.img" ] || {
92    msg "Generating base.img:"
93
94    cmds="
95    # Mount build.img
96    mount -o ro /dev/disk/by-label/build /mnt || exit 1
97
98    # Install some debugging tools
99    apt-get --yes install vim dnsutils iproute2 || exit 1
100
101   # Generate CA root and DNS key+certificate
102   mkdir -p /keys || exit 1
103   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=root.$domain' || exit 1
  LD_LIBRARY_PATH=/mnt/openssl/lib64 /mnt/openssl/bin/openssl req \\
    -newkey ec -pkeyopt ec_paramgen_curve:secp384r1 -nodes \\
    -keyout /keys/$dns_host.key -out /keys/$dns_host.csr -subj
    ↪  '/CN=$dns_host.$domain' || exit 1
  LD_LIBRARY_PATH=/mnt/openssl/lib64 /mnt/openssl/bin/openssl x509 -req \\
    -CA /keys/root.crt -CAkey /keys/root.key -days 3650 -CAcreateserial \\
    -extfile <(printf 'subjectAltName=DNS:$dns_host.$domain,IP:$dns_ip') \\
    -in /keys/$dns_host.csr -out /keys/$dns_host.crt || exit 1
  chmod +r /keys/{root,$dns_host}.key || exit 1"

  for server_cfg in $server_cfgs; do IFS=, read host _ ip _ sites <<< $server_cfg
    cmds="$cmds
    # Generate $host WireGuard and ECH keypair
    mkdir -p /keys/$host || exit 1
    wg genkey | tee /keys/$host/wg.key | wg pubkey > /keys/$host/wg.key.pub || exit 1
    LD_LIBRARY_PATH=/mnt/openssl/lib64 /mnt/openssl/bin/openssl ech \\
      -public_name $host.$domain -pemout /keys/$host/key.ech"
    for site in ${sites//,/ }; do
      cmds="$cmds
      # Generate $site.$domain key+certificate
      LD_LIBRARY_PATH=/mnt/openssl/lib64 /mnt/openssl/bin/openssl req \\
        -newkey ec -pkeyopt ec_paramgen_curve:secp384r1 -nodes \\
        -keyout /keys/$host/$site.key -out /keys/$host/$site.csr -subj
        ↪  '/CN=$site.$domain' || exit 1
      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:$site.$domain,IP:$ip') \\
        -in /keys/$host/$site.csr -out /keys/$host/$site.crt || exit 1
      chmod +r /keys/$host/$site.key || exit 1"
    done
  done

  cmds="$cmds
  # Graceful shutdown
  cd && umount /mnt || exit 1
  shutdown now"

  echo "$cmds"
  debvm-create -h base -o "$dir/base.img" -r unstable -z 1GB -k "$dir/ssh.key.pub"
  ↪  -- \
    --include ca-certificates,wireguard,libpsl5,libpcre3,libnghttp2-14 || exit 1
  debvm-run --image "$dir/base.img" --sshport 2222 --graphical -- \
    -display none -drive file="$dir/build.img",format=raw,if=virtio,readonly=on &
  debvm-waitssh 2222 || exit 1
```

```
  ssh -o NoHostAuthenticationForLocalhost=yes -i "$dir/ssh.key" -p 2222
  ↪ root@127.0.0.1 "$cmds" || exit 1
  wait
}

# Set DNS server configuration
cmds="
# Install dependencies
apt-get --yes install bind9 || exit 1

# Configure BIND9 for DoH usage
>/etc/bind/named.conf.options echo '
tls tlspair {
  key-file \"/keys/$dns_host.key\";
  cert-file \"/keys/$dns_host.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; };
};' || exit 1

# Configure BIND9 to be dns.example.com
>/etc/bind/named.conf.local echo '
zone \"$domain\" {
  type master;
  file \"/var/lib/bind/db.$domain\";
};' || exit 1

# Configure BIND9 with RRs for dns.example.com
>/var/lib/bind/db.$domain echo '
\$TTL 60
@ IN SOA $dns_host root.$dns_host 2007010401 3600 600 86400 600
@ IN NS $dns_host
$dns_host IN A $dns_ip"
for server_cfg in $server_cfgs; do IFS=, read host _ ip _ sites <<< $server_cfg
  cmds="$cmds"$'\n'"$host.ech IN A $ip"
  for site in ${sites//,/ }; do
    for p_server_cfg in $server_cfgs; do IFS=, read p_host _ <<< $p_server_cfg
      [ "$host" != "$p_host" ] && {
        cmds="$cmds"$'\n'"$site IN HTTPS 1 $p_host.ech ech='\$(tail -2
        ↪ /keys/$p_host/key.ech | head -1)'"
      }
```

```
    done
  done
done
cmds="$cmds' || exit 1"

declare "${dns_host}_cmds=$cmds"

# Set TLS servers configuration
for server_cfg in $server_cfgs; do IFS=, read host _ ip wg sites <<< $server_cfg
  cmds="
  # Install dependencies
  apt-get --yes install wireguard tcpdump || exit 1

  # Configure WireGuard
  >/etc/systemd/network/00-wg0.netdev echo '
  [NetDev]
  Name=wg0
  Kind=wireguard
  [WireGuard]
  ListenPort=51820
  PrivateKey='\"\$(cat /keys/$host/wg.key)\"'"
  for p_server_cfg in $server_cfgs; do IFS=, read p_host _ p_ip p_wg _ <<<
  ↪ $p_server_cfg
    [ "$host" != "$p_host" ] && {
      cmds="$cmds
      [WireGuardPeer]
      PublicKey='\"\$(cat /keys/$p_host/wg.key.pub)\"'
      AllowedIPs=$p_wg/32
      Endpoint=$p_ip:51820"
    }
  done
  cmds="$cmds' || exit 1
  >/etc/systemd/network/00-wg0.network echo '
  [Match]
  Name=wg0
  [Network]
  Address=$wg/24' || exit 1

  # Configure NGINX
  mkdir -p /site/nginx/logs || exit 1
  >/site/nginx.conf echo '
  pid /run/nginx.pid;
  worker_processes 1;
  events { worker_connections 1024; }

  # ECH client-facing server as proxy for each WireGuard peer
```

```
  stream {
    log_format basic \"\$remote_addr [\$time_local] \$protocol \$status \$bytes_sent
    ↪  \$bytes_received \$session_time\";
    access_log logs/access.log basic;
    ssl_preread on;
    ssl_echkeydir /keys/$host;
    server { listen $ip:443; proxy_pass \$backend; }
    map \$ssl_preread_server_name \$backend {"
  for p_server_cfg in $server_cfgs; do IFS=, read _ _ _ p_wg p_sites <<<
  ↪  $p_server_cfg
    for p_site in ${p_sites//,/ }; do
      cmds="$cmds $p_site.$domain $p_wg:443;"
    done
  done
  cmds="$cmds
    }
  }

  # ECH backend server listening only through WireGuard
  http {"
  for site in ${sites//,/ }; do
    cmds="$cmds
    server {
      root /site/$site;
      server_name $site.$domain;
      listen $wg:443 ssl;
      http2 on;
      ssl_certificate /keys/$host/$site.crt;
      ssl_certificate_key /keys/$host/$site.key;
      ssl_protocols TLSv1.3;
      location / { ssi on; index index.html index.htm; }
    }"
  done
  cmds="$cmds
  }' || exit 1"

  for site in ${sites//,/ }; do
    cmds="$cmds
    # Generate $site index.html
    mkdir -p /site/$site || exit 1
    >/site/$site/index.html echo '\
    <!doctype html>
    <html lang=en>
      <head>
        <meta charset=utf-8>
        <title>$site.$domain</title>
```

```
      </head>
      <body>
        <img src=\"/image.png\" width=\"300\" height=\"300\">
        <p>
          Welcome to <b>$site.$domain</b><br/>
          Got here via <i><!--# echo var=\"remote_addr\" --></i>
        </p>
        <ul>
          <li>SNI: <!--# echo var=\"ssl_server_name\" --></li>
          <li>HTTP host: <!--# echo var=\"http_host\" --></li>
          <li>ALPN protocol: <!--# echo var=\"ssl_alpn_protocol\" --></li>
        </ul>
        <form action=\"/pkglist\">
          <input type=\"submit\" value=\"Download pkglist\" />
        </form>"
    for p_server_cfg in $server_cfgs; do IFS=, read p_host _ p_ip _ p_sites <<<
    ↪  $p_server_cfg
      cmds="$cmds<p>Sites on $p_host ($p_ip):"
      for p_site in ${p_sites//,/ }; do
        cmds="$cmds<br/><a href=\"https://$p_site.$domain\">$p_site.$domain</a>"
        [ "$site" = "$p_site" ] && cmds="$cmds *" || true
      done
      cmds="$cmds</p>"
    done
    cmds="$cmds
      </body>
    </html>' || exit 1
    ln -s /mnt/src/openssl/doc/images/openssl-square-nontransparent.png
    ↪  /site/$site/image.png || exit 1
    ln -s /var/lib/apt/lists/deb.debian.org_debian_dists_unstable_main_binary-amd64_
    ↪  Packages /site/$site/pkglist || exit 1"
  done

  cmds="$cmds
  # WireGuard traffic padding service
  >/site/padding.sh echo '#!/bin/bash
  tc qdisc replace dev enp0s6 root netem slot 100ms 200ms
  tcpdump -i wg0 -nnqlt ip and src $wg and not dst port 12345 | while read _ _ _ dst
  ↪  _ len; do"
  for p_server_cfg in $server_cfgs; do IFS=, read _ _ _ p_wg _ <<< $p_server_cfg
    cmds="$cmds
    [ \"$p_wg\" != \"\${dst%.*}\" ] && dd status=none if=/dev/urandom bs=\$len
    ↪  count=1 >/dev/udp/$p_wg/12345 &"
  done
  cmds="$cmds
  done' || exit 1
```

```
  >/site/padding.service echo '
  [Unit]
  After=network-online.target
  [Service]
  ExecStart=/site/padding.sh
  Restart=always
  [Install]
  WantedBy=multi-user.target' || exit 1
  chmod +x /site/padding.sh || exit 1

  # Install services
  cp /site/padding.service /mnt/nginx/nginx.service /etc/systemd/system || exit 1
  systemctl daemon-reload && systemctl enable padding nginx || exit 1"

  declare "${host}_cmds=$cmds"
done

# Generate all VM images in parallel
port=2222
for cfg in "$dns_host,$dns_mac,$dns_ip" $server_cfgs; do IFS=, read host mac ip _
↪  <<< $cfg
  port="$((port+1))"
  [ -f "$dir/$host.img" ] || {
    msg "Generating $host.img:"

    cmds_var="${host}_cmds"
    cmds="
    # Mount build.img
    >>/etc/fstab echo 'LABEL=build /mnt ext4 defaults 0 0' || exit 1
    mount -o ro /dev/disk/by-label/build /mnt || exit 1

    # Useful aliases
    >~/.profile echo '
    alias openssl=\"LD_LIBRARY_PATH=/mnt/openssl/lib64 /mnt/openssl/bin/openssl\"
    alias curl=\"LD_LIBRARY_PATH=/mnt/openssl/lib64 /mnt/curl/bin/curl\"
    echo \"dig +https @dns.example.com tcd.example.com https\"
    echo \"curl --verbose --cacert /keys/root.crt --ech hard --doh-url
    ↪  https://dns.example.com/dns-query https://tcd.example.com\"
    ' || exit 1

    # Configure networking
    hostnamectl set-hostname $host || exit 1
    sed -i 's/base/$host/g' /etc/hosts || exit 1
    >/etc/systemd/network/00-br0.network echo '
    [Match]
    MACAddress=$mac
```

```
    [Network]
    DNS=$dns_ip
    Address=$ip/24
    [Route]
    Gateway=0.0.0.0
    Destination=0.0.0.0/0
    Metric=9999' || exit 1

    # Execute $host-specific commands
    ${!cmds_var}

    # Graceful shutdown
    cd && umount /mnt || exit 1
    shutdown now"

    echo "$cmds"
    cp "$dir/base.img" "$dir/$host.img" || exit 1
    debvm-run --image "$dir/$host.img" --sshport "$port" --graphical -- \
      -display none -drive file="$dir/build.img",format=raw,if=virtio,readonly=on &
    debvm-waitssh "$port" || exit 1
    ssh -o NoHostAuthenticationForLocalhost=yes -i "$dir/ssh.key" -p "$port"
    ↪  root@127.0.0.1 "$cmds" || exit 1
  } &
done
wait

port=2222
for cfg in "$dns_host,$dns_mac" $server_cfgs; do IFS=, read host mac _ <<< $cfg
  sleep 1
  port="$((port+1))"
  {
    msg "Booting up host $host..."
    debvm-run --image "$dir/$host.img" --sshport "$port" --graphical -- \
      -display none -drive file="$dir/build.img",format=raw,if=virtio,readonly=on \
      -device virtio-net-pci,netdev=net1,mac=$mac -netdev bridge,id=net1,br=br0 &
    debvm-waitssh "$port" || exit 1
    msg "Host $host is up and running"
    echo "ssh -o NoHostAuthenticationForLocalhost=yes -i '$dir/ssh.key' -p $port
    ↪  root@127.0.0.1"
    wait
    msg "Host $host has shutdown"
  } &
done
wait
killall debvm-run qemu-system-x86_64
```

A2 ECH-enabled curl output

<explanation>

```
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
```