Towards Forward Secure Internet Traffic

Eman Salem Alashwali[1,2], Pawel Szalachowski[3], and Andrew Martin[1]

[1] University of Oxford, Oxford, United Kingdom
{eman.alashwali,andrew.martin}@cs.ox.ac.uk
[2] King Abdulaziz University (KAU), Jeddah, Saudi Arabia
[3] Singapore University of Technology and Design (SUTD), Singapore, Singapore
pawel@sutd.edu.sg

Abstract. Forward Secrecy (FS) is a security property in key-exchange algorithms which guarantees that a compromise in the secrecy of a long-term private-key does not compromise the secrecy of past session keys. With a growing awareness of long-term mass surveillance programs by governments and others, FS has become widely regarded as a highly desirable property. This is particularly true in the TLS protocol, which is used to secure Internet communication. In this paper, we investigate FS in pre-TLS 1.3 protocols, which do not mandate FS, but still widely used today. We conduct an empirical analysis of over 10 million TLS servers from three different datasets using a novel heuristic approach. Using a modern TLS client handshake algorithms, our results show 5.37% of top domains, 7.51% of random domains, and 26.16% of random IPs *do not select* FS key-exchange algorithms. Surprisingly, 39.20% of the top domains, 24.40% of the random domains, and 14.46% of the random IPs that *do not select* FS, *do support* FS. In light of this analysis, we discuss possible paths toward forward secure Internet traffic. As an improvement of the current state, we propose a new client-side mechanism that we call "Best Effort Forward Secrecy" (BEFS), and an extension of it that we call "Best Effort Forward Secrecy and Authenticated Encryption" (BESAFE), which aims to guide (force) misconfigured servers to FS using a best effort approach. Finally, within our analysis, we introduce a novel adversarial model that we call "discriminatory" adversary, which is applicable to the TLS protocol.

1 Introduction

1.1 Problem

Forward Secrecy (FS) is a security property in key-exchange algorithms which guarantees that a compromise in the secrecy of a long-term private-key does not compromise the secrecy of past session keys [25]. With a growing awareness of long-term mass surveillance programs by governments and others, FS has become widely regarded as a highly desirable property. This is particularly true in the TLS protocol, which is used to secure Internet communication. Experience has shown the possibility of servers' long-term private-key compromise. For example, RSA [34] long-term private-keys have been compromised through prime factorisation, due to advancement in computing power [10][20], or due to low entropy during keys generation [16]. Furthermore, long-term private-keys can be compromised through implementation bugs as

in the Heartbleed bug [39], through social engineering, or other attacks. Due to the increasing importance of FS, the new version of TLS, TLS 1.3, mandates it by design by prohibiting non-FS key-exchange algorithms [33]. In recent years, it has been shown that some FS key-exchange algorithms (e.g. ECDHE) can achieve faster performance than non-FS (e.g. RSA) algorithms [18]. Despite recommendations to server administrators to select FS key-exchange algorithms, non-FS key-exchange algorithms are selected by more than 25% of the servers in our IPs dataset as we will show later. As a result, clients proceed with non-FS key-exchange algorithms when connecting to these servers. This puts users' encrypted data at the risk of future decryption by adversaries who collect traffic today, and decrypt it whenever the targeted servers' private-key is compromised. Motivated by the importance of FS in Internet security, in this paper, we analyse the state of FS in pre-TLS 1.3 protocols, and discuss possible paths towards improving its adoption, including proposing a new best effort approach.

1.2 Contribution

Our contributions are as follows: first, we conduct an empirical analysis of FS on over 10 million TLS servers using a novel heuristic approach on three different datasets that contain top domains, random domains, and random IPs, which represent the real-world web. Unlike previous work that identifies servers that select non-FS key-exchange algorithms by capturing the servers' responses for TLS handshakes, our analysis employs a heuristic procedure that allows us to answer a deeper question: *Do servers that select non-FS key-exchange algorithms support FS ones?* Our results provide new and useful insights to vendors, policy makers, and decision makers. Second, we discuss possible paths towards forward secure Internet traffic. Third, we propose a novel client-side mechanism that we call "Best Effort Forward Secrecy" (BEFS), and an extension of it that we call "Best Effort Forward Secrecy and Authenticated Encryption" (BESAFE), which aims to guide (force) misconfigured servers to FS key-exchange algorithms using a best effort approach. We implement and evaluate a proof-of-concept for it. Our mechanism adds value to the existing "all or nothing" approach. Finally, within our BEFS security analysis, we introduce a novel threat model that we call "discriminatory" adversary. The model is applicable to semi-trusted servers running protocols such as TLS that gives the server the power of selecting a security level, exemplified by the ciphersuite in our case, while the client has no means of verifying the server's actual capabilities (i.e. justifying the server's decision if it selects a non-preferred ciphersuite such as those that do not provide FS). We show how this power can be abused by semi-trusted servers to discriminate against their users for a powerful third-party's advantage (e.g. government intelligence), with minimal evidence and liabilities (e.g. legal) of the server's involvement in carrying out the attack.

1.3 Scope

Our focus is pre-TLS 1.3 protocols, mainly the currently supported versions by most clients and servers, TLS 1.2, TLS 1.1, and TLS 1.0. As a shorthand, we refer to them as pre-TLS 1.3. TLS 1.2 [32] does not enforce FS by design, but still widely used today. As of April 2019, only 13.6% of the top 150,000 most popular domains support TLS 1.3, according to a report by SSL Labs [31]. Furthermore, there are no known plans from

standardisation bodies or browser vendors to deprecate TLS 1.2 yet. Although our work is mainly on pre-TLS 1.3, it has an impact on currently deployed systems.

2 Background

2.1 Transport Layer Security (TLS)

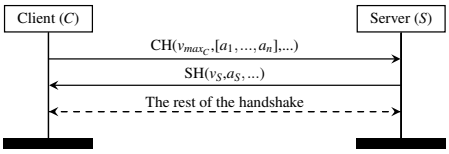

Fig. 1: Illustration of the version and ciphersuite negotiation in pre-TLS 1.3 protocols.

Transport Layer Security (TLS) [33][32] is one of the most important and widely used protocols to date. It is the main protocol used to secure Internet communication. TLS aims to provide data confidentiality, integrity, and authentication between two communicating parties. It has been in use since 1995, and was formerly known as the Secure Socket Layer (SSL). TLS consists of multiple sub-protocols including the TLS handshake protocol. In the handshake protocol, both communicating parties authenticate each other and negotiate security-sensitive parameters, including the protocol version and ciphersuite. The ciphersuite is an identifier that defines the cryptographic algorithms that, upon agreement between the communicating parties (client C and server S), will be used to secure subsequent messages of the protocol. In pre-TLS 1.3, the ciphersuite defines the key-exchange, authentication, symmetric encryption, and hash algorithms. Some ciphersuites provide stronger security properties than others. For example, FS guarantees that a compromise in the server's long-term private-key does not compromise past session keys [25]. Similarly, Authenticated Encryption (AE) provides confidentiality, integrity, and authenticity simultaneously, which provides stronger resilience against some attacks over the MAC-then-encrypt schemes [40][6]. Most TLS clients today, such as mainstream web browsers, offer a mixture of ciphersuites that provides various levels of security such as FS, AE, both FS and AE, or none of them. The same applies to servers that select the session's ciphersuite.

As depicted in Figure 1, at the beginning of a new TLS handshake, both communicating parties negotiate and agree on a protocol version and ciphersuite. The client sends a `ClientHello` (`CH`) message to the server. The `CH` contains several parameters including the client's maximum supported version v_{max_C} and a list of ciphersuites $[a_1, ..., a_n]$ (we refer to them as the **client's offered versions** and the **client's offered ciphersuites**). Upon receiving the `CH`, the server selects a single version v_S and a ciphersuite a_S from the client's offer (we refer to them as the **server's selected version** and the **server's selected sciphersuite**), and responds with a `ServerHello` (`SH`) containing v_S and a_S. If the server does not support the client's offered versions or ciphersuites, i.e. the client's offer is *not* in the **server's supported versions** or the **server's supported ciphersuites**, the server responds with a handshake failure alert.

2.2 TLS Key-Exchange Algorithms

There are two main key-exchange algorithms used in pre-TLS 1.3 protocols: the Rivest-Shamir-Adleman (RSA) [34], and the Ephemeral Diffie-Hellman (DHE) [11]. DHE has two variants: the Finite-Field (DHE) and the Elliptic-Curve (ECDHE). We use the term (EC)DHE to refer to either ECDHE or DHE.

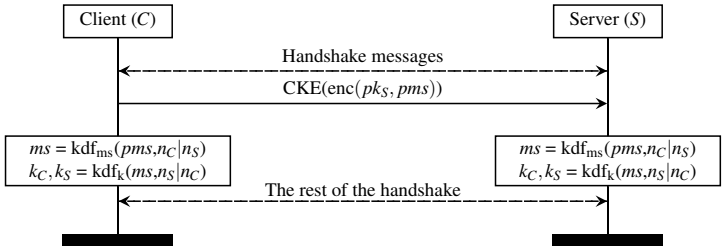

Fig. 2: Illustration of the RSA key-exchange in pre-TLS 1.3.

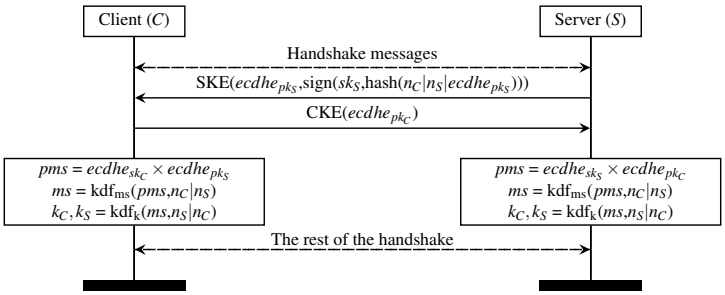

Fig. 3: Illustration of the (EC)DHE key-exchange in pre-TLS 1.3.

RSA Key-Exchange [34] does not guarantee FS. As depicted in Figure 2, to generate a session key using RSA, the client generates a random value for the pre-master secret *pms*, encrypts it with the server's long-term RSA public-key pk_S using the (enc) function, then sends it in a `ClientKeyExchange (CKE)` message. After that, both parties derive the master secret *ms* from the *pms* and their nonces n_C and n_S, using a Key Derivation Function (kdf_{ms}). Then, they compute the session keys k_C and k_S using the (kdf_{k}) function. Clearly, the secrecy of the *pms* relies on the secrecy of the server's long-term private-key sk_S that is associated with the server's public-key pk_S since every *pms* is encrypted with the same server's long-term key pk_S during the key's lifetime. Therefore, if the server's long-term private-key sk_S is compromised at some point in the future, a passive adversary who has been collecting encrypted traffic, can recover the *pms*, and consequently, the *ms*, k_C and k_S, and hence decrypt past sessions' encrypted data.

(EC)DHE [28][11] guarantees FS. As depicted in Figure 3, to generate a session key using (EC)DHE, the server sends its (EC)DHE public-key parameters $ecdhe_{pk_S}$, signed with its long-term private-key sk_S using the (sign) function in a `ServerKeyExchange` (`SKE`) message. The client then sends its (EC)DHE public-key parameter $ecdhe_{pk_C}$ in a `ClientKeyExchange` (`CKE`) message. After that, both parties compute their *pms*, derive the *ms* using the (kdf_{ms}) function. Then, they compute the session keys k_C and k_S using

the (kdf_k) function. The (EC)DHE key is ephemeral, i.e. a fresh key is generated for each session. Unlike RSA, in (EC)DHE key-exchange, the *pms* is not encrypted with the server's long-term key pk_S. Therefore, the *ms*, k_C and k_S, do not rely on the secrecy of the server's long-term private-key sk_S. Hence, if the server's long-term private-key sk_S is compromised at some point in the future, a passive adversary who has been collecting encrypted traffic, cannot recover the *ms*, k_C, and k_S of past sessions.

2.3 Terminology

Throughout the paper, we use the term **FS-ciphersuites** to denote ciphersuites that support FS using the ECDHE key-exchange algorithm. We use the term **AE-ciphersuites** to denote ciphersuites that support AE using either the ChaCha20 stream cipher or the GCM mode of operation in the symmetric encryption algorithm. We use the term **FS+AE-ciphersuites** to denote ciphersuites that support both FS and AE using the ECDHE key-exchange algorithm and either the ChaCha20 stream cipher or the GCM mode of operation. Properties preceded with a "non" denotes a negated property. For example, the term **FS+non-AE-ciphersuites** denotes ciphersuites that support FS but not AE. These definitions are not meant for generalisation. They are limited to the paper's scope and to our experiment settings which are based on Google `Chrome`'s[1] ciphersuites. For example, `Chrome` only supports ECDHE to provide FS. Hence, our definition of FS-ciphersuites considers ECDHE only. To describe domains, we use the terms we defined in [4]: **main-domains** denotes domains consisting of a Top Level Domain (TLD) (e.g. "com") prefixed by a single label, and do not have any further sub-domains, e.g. "example.com"; **plain-domains** denotes domains that are not prefixed with "www" sub-domains, e.g. "example.com"; and **www-domains** denotes domains that are prefixed with "www" sub-domains, e.g. "www.example.com".

3 Empirical Study

3.1 Datasets

We build three datasets that we name: `top-domains`, `random-domains`, and `random-ips`. We end up with 999,884 distinct domains in the `top-domains` dataset, 4,960,390 distinct domains in the `random-domains` dataset, and 4,881,985 distinct IPv4 addresses in the `random-ips` dataset. The rationale behind choosing these three categories is to represent the real-world web as much as possible. In what follows, we explain how we build and pre-process each dataset.

Top Domains Dataset The `top-domains` dataset initial size is 1 million domains, obtained from the `Alexa` list of top 1 million most visited domains globally [5], retrieved on Aug. 22, 2018. We exclude the www-domains because we target plain-domains (see section 2.3 for our definitions of plain-domains and www-domains), which are the majority in the `Alexa` list. After excluding the www-domains, we end up with 999,884 domains that are mainly (around 94.81%) classified as main-domains.

Random Domains Dataset The `random-domains` dataset initial size is 5 million random domains obtained from a large dataset that contains 54,063,220 distinct (alphabetically unordered) domains that successfully completed a TLS handshake in Amann et al. [7],

[1] As a shorthand, throughout the paper, we refer to Google `Chrome` as `Chrome`

which have been collected from multiple sources. To maintain consistency with the top domains dataset format, we extract 5 million domains from [7] that are classified as both plain-domains and main-domains. In this dataset, the TLDs scope is limited to generic TLDs (gTLDs), and does not include "multi-level" TLDs such as country-code TLDs (ccTLDs), e.g. "ac.uk". This is to avoid the complexity of distinguishing domains that have sub-domains from domains that have ccTLDs, which is somewhat difficult to achieve with 100% accuracy. To avoid repeated domains, from the 5 million random domains, we exclude the domains that exist in the top domains dataset, either "as is" or as a main-domain of a sub-domain in the top domains dataset (the top domains dataset contains a small percentage of sub-domains). We identify sub-domains in the top domain dataset in two steps: first, by using a regular expression, we extract the domains that have more than one dot ".". Second, with the aid of `tldextract` [22] python library, we distinguish sub-domains from domains with country-code TLDs (ccTLD) such as "example.ac.uk" (the latter is considered a main-domain). We end up with 4,960,390 distinct random domains.

Random IPs Dataset The `random-ips` dataset initial size is 5 million distinct IPv4 addresses that have completed a successful TLS handshake with `Censys`, a search engine and database for servers and network devices on the Internet [13], retrieved from the `Censys` IPv4 dataset on Oct. 20, 2018, through research access to the `Censys` database. To avoid repeated IPs, from the 5 million IPs, we exclude the IPs that are associated with any domain that has responded to a handshake in the scanning or inspection phases (further details on the scanning and inspection phases will be provided in the methodology in section 3.3). For this reason, we build the random IPs dataset after we finish the domains datasets scanning and inspection phases. We end up with 4,881,985 IPs.

3.2 Research Questions

Our analysis aims to answer the following main questions:

1. What is the percentage of servers that select non-FS-ciphersuites today?
2. Do servers that select non-FS-ciphersuites support FS-ciphersuites?
3. Do different dataset natures result in different trends in selecting and supporting FS-ciphersuites?

Whilst addressing these main questions, the following side questions arose:

1. What is the percentage of servers that select FS+non-AE-ciphersuites after the client's FS-ciphersuites enforcement[2]?
2. Do servers that select FS+non-AE-ciphersuites after the client's FS-ciphersuites enforcement support FS+AE-ciphersuites?
3. Can the client's FS-ciphersuites enforcement lead servers to lose the AE property?
4. Do servers that lose the AE property after the client's FS-ciphersuites enforcement support FS+AE-ciphersuites?

[2] The term "client's FS-ciphersuite enforcement" refers to a client that offers FS-ciphersuites exclusively. The same applies for "client's FS+AE-ciphersuite enforcement" but the latter offers FS+AE-ciphersuites exclusively. More details are provided next in the methodology in section 3.3

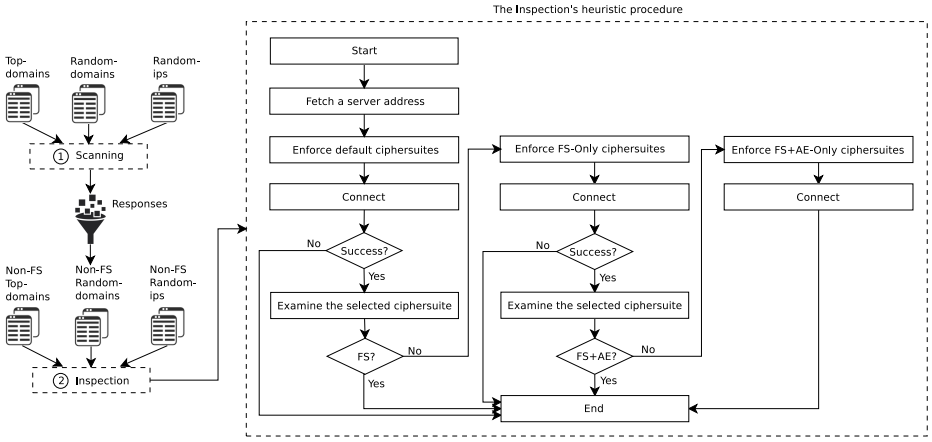

Fig. 4: A general overview of our methodology showing the two phases and their input.

3.3 Methodology

As depicted in Figure 4, our methodology consists of two main phases: a scanning phase followed by an inspection phase.

Scanning Phase We consider the scanning phase as an exploration phase. In this phase, for each server address in our datasets, we perform a TLS handshake using the `tls-scan` tool [43], an open source fast TLS scanner capable of performing concurrent TLS connections. We customise the `tls-scan` to utilise the `OpenSSL 1.1.0g` library, and to support `Chrome`'s latest version pre-TLS 1.3 ciphersuites, which support various ciphersuites that provide FS, or AE, or none of them. We choose to base the scanning client on `Chrome`'s ciphersuites because `Chrome` is the most representative TLS client on the Internet. As of Feb. 2019, `Chrome`'s usage is 79.7% [41]. `tls-scan` includes the Server Name Indication (SNI)[3] extension for domain name scans by default. We set the timeout argument to 5 seconds, and the concurrency argument to 50 connections. We ran the scans between Aug. 23, 2018 and Oct. 21, 2018 at the University of Oxford in discrete intervals based on the dataset.

Inspection Phase After the scanning phase is complete, we extract the responding addresses that selected non-FS-ciphersuites in the scanning phase. Each dataset is inspected within a maximum of 48 hours after its scanning phase is complete. For the inspection phase, we develop a TLS client that implements our heuristic procedure (see Figure 4), which works as follows, for each server's address:

1. The client performs a TLS handshake based on `Chrome`'s pre-TLS 1.3 default ciphersuites. This first handshake of the inspection phase is similar to the scanning

[3] The SNI extension passes the domain name in the TLS handshake in order to obtain more accurate responses in virtual hosting environments, where a single server can host multiple domains [14].

phase handshake[4]. The inspection's first handshake serves as a confirmation of the server's selected ciphersuite. It records the server's selected ciphersuite from a default client's view just before the heuristic procedure starts. If the handshake failed, the client records the error, and the heuristic procedure ends here.

2. Upon receiving the server's response to the first handshake (step 1), the client checks the server's selected ciphersuite: if it is a FS-ciphersuite, this means that the server has changed its behaviour after the scanning phase since all the inspection input addresses are for servers that selected non-FS-ciphersuites in the scanning phase. The client records the server's response, and the heuristic procedure for this server ends here. Otherwise, if the server's selected ciphersuite is still a non-FS-ciphersuite, we classify this server as "**stable**", i.e. consistently selects a non-FS-ciphersuite. The client then updates its TLS context to support FS-ciphersuites *exclusively*. The set of FS-ciphersuites may or may not support AE, i.e. it contains FS+AE-ciphersuites and FS+non-AE-ciphersuites. This context is more restricted than the default one.
3. The client then performs a second handshake utilising the new FS-ciphersuites context. If the handshake failed, the client records the error and the heuristic procedure for this server ends here.
4. Upon receiving the server's response to the second handshake (step 3), the client checks the server's selected ciphersuite: if it is a FS+AE-ciphersuite, this means that the server supports FS+AE-ciphersuite after the client's FS-ciphersuites enforcement. The client then records the server's response, and the heuristic procedure for this server ends here. Otherwise, if the server's selected ciphersuite is a FS+non-AE-ciphersuite, the client updates its context to support FS+AE-ciphersuites *exclusively*. This context is more restricted than the FS-ciphersuites context.
5. The client then performs a third handshake utilising the new FS+AE-ciphersuites context. If the handshake failed, the client records the error and the heuristic procedure ends here.
6. Upon receiving the server's response, the client records the response. The heuristic procedure for this server ends here.

We develop and run the inspection client using `python 3.6.5`. Similar to the `tls-scan` client in the scanning phase, it utilises `OpenSSL 1.1.0g`. We enable the SNI for the top and random domains inspection (the IPs dataset do not need the SNI), and we set the timeout to 5 seconds. The results are then stored and analysed using MySQL database and queries.

Identifying Device Types We classify device types into two categories: ordinary web servers and network devices. We use the term "**network device**" to refer to non-ordinary TLS servers, e.g. embedded web servers in network devices such as routers. To identify the device type, we input the IPs of the dataset in question in a query to `Censys` database to get the IPs metadata. We then produce a breakdown of the responding IPs grouped by the device type. We base our device types queries on the IPs, i.e. in the domains datasets, we first extract the distinct IPs behind the domains, because `Censys` is mainly an IP-based engine. We query the `Censys` snapshot that dates to the starting date of the

[4] Except that our inspection client does not support SSL v3 while the scanning `tls-scan` client supports SSL v3. However, we analyse FS regardless of the client's supported versions.

Table 1: Summary of the scanning results. Every additional indentation means that the percentages are computed out of the previous level results. The "% Network devices" are computed over the responding IPs to `Censys` metadata query (exact numbers are provided in text).

	Datasets		
	`top-domains`	`random-domains`	`random-ips`
Dataset size	999,884	4,960,390	4,881,985
Responding servers	814,333 (81.44%)	3,221,249 (64.94%)	4,477,279 (91.71%)
Distinct IPs	468,346 (57.51%)	690,912 (21.45%)	4,477,279 (100%)
% Network devices	466 (0.10%)	1208 (0.18%)	518,988 (11.59%)
Select non-FS	43,756 (5.37%)	241,994 (7.51%)	1,171,101 (26.16%)

scan or inspection (depending on the phase) of the dataset in question. `Censys` labels the device type of the network devices that it identifies, e.g. "DSL/cable modem". If the device type field is empty, this means that the device is either an ordinary web server, or a network device that `Censys` cannot identify. Finally, we do not always obtain 100% responses for the IPs that we query their metadata from `Censys`. However, overall, the percentages of the responses that we receive are between 98.36% to 100% (depending on the dataset) of the IPs we query.

Ethical Considerations Our study is in line with the ethical recommendations in carrying out measurement studies [29]. First, we do not collect private data. Second, we do not perform an exhaustive number of handshakes on any single server. Our clients' handshakes can by no means be classified as a Denial of Service (DoS) attack. Third, we use a designated public IPv4 address per scanning device instead of Network Address Translation (NAT), to avoid potential disturbance to other users in our institution's network if a server has blocked our scanning or inspection device's IP. Fourth, we use informative DNS names that contain "TLS probing" to help server administrators identify our devices' activity in their logs. Finally, we inform the IT and security teams in our institution where the empirical study has been conducted so they expect a high volume of outgoing connections from our experiment devices, and to expect some incoming blacklisted certificates from random servers.

3.4 Results

Scanning Phase In this phase, we input the servers' addresses in our datasets. The results of the scanning phase are summarised in Table 1.

Responding servers As illustrated in Table 1, the highest percentage of responses is in random IPs (91.71%), followed by top domains (81.44%), and finally random domains (64.94%). The response rate is influenced by the dataset category. Both the IPs and top domains datasets are recent. That is, the addresses in the IPs dataset have recently completed a TLS handshake with the `Censys` engine [13], and TLS adoption in top domains is high. The low response rate in random domains (64.94%) is very likely due to the dataset age. It is obtained from a previous study that was published in 2017 [7]. Hence, many domains could have gone down since then.

Table 2: Summary of the inspection results. Every additional indentation means the percentages are computed out of the previous level results. The input of the inspection phase is the servers that selected non-FS-ciphersuites in the scanning phase. The "% Network devices" are computed over the responding IPs to `Censys` metadata query (exact numbers are provided in text).

	Datasets		
	`non-FS top-domains`	`non-FS random-domain`	`non-FS random-ips`
Dataset size	43,756	241,994	1,171,101
Responding servers	43,374 (99.13%)	240,519 (99.39%)	1,111,802 (94.94%)
Select non-FS (stable)	43,158 (99.50%)	240,274 (99.90%)	1,111,174 (99.94%)
Distinct IPs	33,474 (77.56%)	61,522 (25.60%)	1,111,174 (100%)
% Network devices	76 (0.23%)	361 (0.59%)	434,076 (39.06%)
Support FS	16,916 (39.20%)	58,636 (24.40%)	160,706 (14.46%)
Distinct IPs	12,545 (74.16%)	13,839 (23.60%)	160,706 (100%)
% Network devices	12 (0.10%)	27 (0.20%)	1503 (0.94%)
Select FS+non-AE	10,091 (59.65%)	38,583 (65.80%)	93,566 (58.22%)
Support FS+AE	1629 (16.14%)	1289 (3.34%)	24,128 (25.79%)
Lose AE	2686 (26.62%)	1768 (4.58%)	12,769 (13.65%)
Support FS+AE	45 (1.68%)	91 (5.15%)	4668 (36.56%)

In terms of device types, from the responding top domains, there are 468,346 (57.51%) distinct IPs behind all the top domains. We receive metadata responses for 464,191 (99.11%) of them from the `Censys` database. Of those, only 466 (0.10%) IPs are labeled as network devices. From the responding random domains, there are 690,912 (21.45%) distinct IPs behind them. We receive metadata responses for 686,085 (99.30%) of them. Of those, there are 1208 (0.18%) labeled as network devices. From the responding random IPs, we receive metadata responses for all of them (100%). Of those, there are 518,988 (11.59%) IPs labeled as networked devices. Clearly, the percentage of network devices in the random IPs is higher than that in the top and random domains.

Servers that select non-FS-Ciphersuites From the responding servers, we find 5.37% of the top domains, 7.51% of the random domains, and 26.16% of the random IPs, select non-FS-ciphersuites. The lowest percentage is in the top domains, the highest is in the random IPs, while in the random domains, it is slightly higher than that in the top domains. The fact that the random IPs dataset has the highest percentage of network devices can be correlated to the high percentage of servers that select non-FS-ciphersuites. We can confirm this in the inspection phase when we look closer at the device types of those servers that select non-FS-ciphersuites.

Inspection Phase In this phase, we input the addresses of servers that select non-FS-ciphersuites in the scanning phase. Table 2 summarises the inspection phase results.

Responding servers As Table 2 illustrates, over 99% of top and random domains, and 94.94% of IPs that select non-FS-ciphersuites in the scanning phase, have responded to our inspection client's handshake. The low response rate in the IPs dataset compared to the top and random domains datasets is very likely attributed to SSL v3 devices as our inspection client does not support SSL v3, while the scanning client does. It is also very likely that those non-responding IPs are mostly for network devices since using legacy versions in network devices is more common than that in ordinary web servers [37].

Servers that still select non-FS-Ciphersuites (stable) In our work, we use the term "**stable**" to refer to servers that consistently select non-FS-ciphersuites in both the inspection's first handshake and the scanning handshake. As shown in Table 2, clearly, the stability in selecting non-FS-ciphersuites among all datasets is high (over 99% in all datasets), despite the difference in the supported protocol versions in the scanning and inspection clients (the scanning client supports SSLv3 while the inspection does not). This suggests that, to some extent, servers' selected ciphersuites are not affected by the negotiated versions.

In terms of device types, out of the top domains that select non-FS-ciphersuites, there are 33,474 (77.56%) distinct IPs behind all these domains. Of those, we receive metadata responses for 33,079 (98.82%) IPs from the `Censys` database. Of those, there are 76 (0.23%) IPs labeled as networked devices. Of the random domains that select non-FS-ciphersuites, there are 61,522 (25.60%) distinct IPs behind them. We receive metadata for 61,176 (99.44%) IPs from `Censys`. Of those, there are 361 (0.59%) IPs labeled as networked devices. Of the random IPs that select non-FS, we receive metadata for 1,111,174 (100%). Of those, there are 434,076 (39.06%) IPs labeled as networked devices. Network devices represent no more than 0.59% of top and random domains that select non-FS-ciphersuites. However, more than a third of servers that select non-FS-ciphersuites in the random IPs dataset are labeled as network devices. The high percentage of network devices in the random IPs is likely the reason for the high percentage of servers that select non-FS-ciphersuites.

Servers that select non-FS-Ciphersuites, but support FS-Ciphersuites We find 39.20% of top domains, 24.40% of random domains, 14.46% of random IPs, that select non-FS-ciphersuites in the inspection phase, do support FS-ciphersuites. The top-domains are the highest, followed by the random domains, and finally, the random IPs are the lowest. Interestingly, this is a shifted paradigm for the percentages of servers that select non-FS-ciphersuites that is shown in Table 1, where the random IPs have the highest percentage and the top domains have the lowest percentage. The results reflect that the lack of FS-ciphersuite selection in the top and random domains is to a large extent due to misconfiguration, while in the random IPs, it is mostly due to lack of support.

In terms of device types, out of the top domains that select non-FS-ciphersuites but support FS-ciphersuites, there are 12,545 (74.16%) distinct IPs behind them. We receive metadata responses for 12,339 (98.36%) IPs. We find 12 (0.10%) of them are labeled as networked devices. Of the random domains that select non-FS-ciphersuites but support FS-ciphersuites, there are 13,839 (23.60%) distinct IPs behind them. We receive metadata responses for 13,744 (99.31%) IPs. Of those, 27 (0.20%) are labeled as network devices. Of the random IPs, that select non-FS-ciphersuites but support FS-ciphersuites, we receive metadata responses for 160,706 (100%) IPs. Of those, 1503 (0.94%) are la-

beled as network devices. The results show that the majority of those devices are not identified as network devices, even in the random IPs dataset that shows the highest percentage of network devices. Those servers that select non-FS-ciphersuites and turned to support FS-ciphersuites are not network devices. Therefore, most of the network devices that select non-FS-ciphersuites, do not support FS-ciphersuites.

Servers that select FS+non-AE-Ciphersuites after enforcing FS-Ciphersuites Out of the top domains, random domains, and random IPs that support FS-ciphersuites after enforcement (row label "Support FS" in Table 2), there are 59.65% top domains, 65.80% random domains, and 58.22% random IPs that select FS+non-AE-ciphersuites. The reason for selecting non-AE can be attributed to the fact that TLS 1.0 and TLS 1.1 do not support AE-ciphersuites [36], and these devices might be running legacy versions of TLS. Otherwise, this is attributed to misconfiguration. To better understand this situation, we next check whether those servers support FS+AE-ciphersuites or not.

Servers that select FS+non-AE-Ciphersuite after enforcing FS-Ciphersuites, but support FS+AE-ciphersuites Of the top domains, random domains, and random IPs that select FS+non-AE-ciphersuites after FS-ciphersuites enforcement (row label "Select FS+non-AE" in Table 2), there are 16.14% top domains, 3.34% random domains, and 25.79% random IPs, that support FS+AE-ciphersuite. At this point of the heuristic procedure, the majority of the IPs do not belong to network devices. The majority of the top and random domains that select FS+non-AE-ciphersuites do not support FS+AE-ciphersuites. However, selecting FS+non-AE-ciphersuites while supporting FS+AE-ciphersuites in the IPs dataset is the highest, which we classify as misconfiguration.

When enforcing FS-Ciphersuite causes losing the AE property Of the top domains, random domains, and randm IPs that select FS+non-AE-ciphersuites after enforcing FS-ciphersuites (row label "Select FS+non-AE" in Table 2), there are 26.62% top domains, 4.58% random domains, and 13.65% random IPs, were selecting AE before enforcing FS-ciphersuites, i.e. were selecting non-FS+AE-ciphersuites. This can be either because they do not support any FS+AE-ciphersuites, or due to misconfiguration. This can be clarified next.

Servers that lose the AE property after enforcing FS-Ciphersuites, but support FS+AE-ciphersuites Out of the top domains, random domains, and random IPs that lose the AE property after enforcing FS (row label "Lose AE" in Table 2), we find 1.68% top domains, 5.15% random domains, and 36.56% random IPs, do support FS+AE-ciphersuites. The results reflect that losing the AE property after enforcing the FS in the top and random domains is to a large extent due to a lack of support for FS+AE-ciphersuites, but in the random IPs, it is mostly due to misconfiguration.

4 Towards Forward Secure Internet Traffic

In this section, we discuss possible paths towards forward secure Internet traffic from a client's perspective. Then, we propose and evaluate a novel client-side mechanism that we call Best Effort Forward Secrecy (BEFS), and an extension of it that we call Best Effort Forward Secrecy and Authenticated Encryption (BESAFE). We choose to focus our discussion and solutions on clients because unlike servers, clients are controlled by few players, e.g. browser vendors. Client-side security enhancement mechanisms

are easier to adopt, as shown by recent adoptions of client-side mechanisms such as Google's Certificate Transparency (CT) [23], and others.

4.1 Deprecating non-FS-Ciphersuites in TLS Clients

The most straight-forward approach towards forward secure Internet traffic is deprecating non-FS-ciphersuites from TLS clients. As a result, these clients will not be able to establish TLS connections with servers that do not support FS-ciphersuites. This is a conservative approach that has been taken by browser vendors and standardisation bodies in the past with some protocol versions and algorithms such as SSL v3 [8] and RC4 [30], after their insecurity has become clear. However, deprecating non-FS-ciphersuites *now* can be more problematic than the case of deprecating SSL v3 and RC4 in 2014 and 2016 respectively. By way of comparison, Lee et al.[5] conducted a survey in 2006 to asses the cryptographic strength of TLS servers [24]. It shows that 98.36% of the surveyed servers support TLS 1.0, the latest version at the time of the study, and 57.17% of the servers support AES encryption, which was in its early years as it was standardised in 2001 [15]. In light of these figures, we speculate that SSL v3 and AES adoption when they were deprecated by most browsers in 2014 and 2016 respectively was over 99%. On the other hand, in our results, we calculate an approximation of the servers that support FS-ciphersuites in each dataset. To this end, we first calculate the number of servers that select non-FS-ciphersuites and do not support FS-ciphersuite which can be derived from Table 2 by calculating ("Select non-FS (stable)"−"Support FS"), and then subtracting those results from the overall responses in Table 1's row label "Responding servers", which gives: 788,091 (96.78%) top domains, 3,039,611 (94.36%) random domains, and 3,526,811 (78.77%) random IPs. Our results are in line with `Censys` Oct. 26, 2018 snapshot figures that show 97.44% of `Alexa`'s top domains, and 77.94% of IPs in all IPv4 space, support FS-ciphersuite. However, our results are more accurate as we include not only servers that *select* FS, but also servers that *support* FS but select non-FS, which can be guided through client's enforcement as we will explain next. In addition, we utilises modern client ciphersuites. On the other hand, `Censys` only measures servers that *select* FS, and utilises somewhat legacy ciphersuites. We conclude that the percentages of servers that support FS-ciphersuites are less than that in RC4 and SSL v3 cases, especially in the IPs datasets. The lack of supporting FS-ciphersuites by those servers can be explained by the fact that until recent years, (EC)DHE key-exchange algorithms have been viewed as resource-exhaustive compared to RSA key-exchange, despite the fact that this argument is no longer true with the ECDHE variant as shown in [18].

4.2 Best Effort Forward Secrecy (BEFS)

Overview The gist of our BEFS mechanism is guiding (forcing) misconfigured servers towards FS-ciphersuites. As explained in section 2.1, in ordinary TLS clients such as web browsers, the client offers default ciphersuites, which includes FS-ciphersuites and

[5] Despite the study's age (conducted in 2006), to the best of our knowledge, [24] is the only study that tried to assess servers' supported ciphersuites prior to deprecating RC4 and SSL v3. Note that identifying the *supported* ciphersuites for a server is different from identifying the *selected* ciphersuite. The former requires multiple handshakes, while the latter requires a single handshake, for each server.

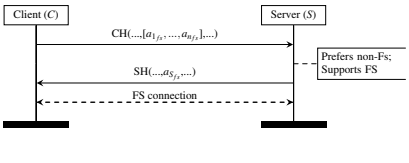

Fig. 5: A BEFS-enabled client handshake when the server prefers to select a non-FS-ciphersuite while supporting FS-ciphersuites. The server is forced to select FS-ciphersuite through client FS-ciphersuite enforcement.

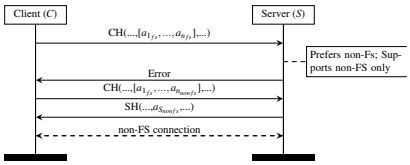

Fig. 6: A BEFS-enabled client handshake when the server does not support FS-ciphersuites. The client falls back to non-FS-ciphersuites only when the server indeed does not support FS.

non-FS-ciphersuites. Upon receiving the client's offer, a server that does not support or does not prefer to select a FS-ciphersuite will select a non-FS-ciphersuite, and sends its selected ciphersuite to the client. The client accepts the server's choice, and the rest of the communication proceeds with a non-FS-ciphersuite. On the other hand, in BEFS, we exploit the TLS ciphersuite negotiation dynamics to influence (bias) the server's choice towards FS-ciphersuites. That is, a BEFS-enabled client first offers FS-ciphersuites *exclusively* $[a_{1_{fs}}, ..., a_{n_{fs}}]$. Upon receiving the client's offered ciphersuites, a server that supports FS-ciphersuites will be guided (forced) to select a FS-ciphersuite $a_{S_{fs}}$, even if it prefers to select a non-FS-ciphersuite, since FS-ciphersuites are the only offered ciphersuites as illustrated in Figure 5. As shown in Table 2, of the servers that select non-FS-ciphersuites, there is between 14.46% to 39.20% that *do support* FS-ciphersuites, which can benefit from the BEFS enforcement mechanism. If the server indeed does not support FS-ciphersuites, it will return a failure alert (see section 2.1 for a background on TLS version and ciphersuite negotiation). In this case, the BEFS-enabled client makes a second handshake utilising default ciphersuites $[a_{1_{fs}}, ..., a_{n_{nonfs}}]$, which includes non-FS-ciphersuites in addition to the previously offered FS-ciphersuites as Figure 6 illustrates. Hence, a server that does not support FS-ciphersuites can still select a non-FS-ciphersuite $a_{S_{nonfs}}$ after the client falls back. BEFS can be viewed as a form of the "Opportunistic Security" concept [12], but at the FS property level. That is, it guides servers to select FS whenever they support it.

The Fallback We now address the fallback aspect. We define three categories of client-side fallbacks: silent fallback, interactive fallback, and signaled fallback. In what follows, we explain them in light of the BEFS mechanism.

Silent fallback. Silent fallbacks do not involve the user or the server. If used in BEFS, if the FS-ciphersuites handshake failed, the client falls back to default ciphersuites (which include non-FS-ciphersuites), in the background, and performs a second handshake utilising default ciphersuites. Silent fallbacks remove the security decision-making overhead from the user at the cost of security. Silent fallbacks do not provide security against active adversaries who can perform downgrade attacks (for a background on downgrade attacks, see [3]). BEFS with silent fallback is secure against passive adversaries, which adds a significant value in the case of FS. It makes mass surveillance

more difficult to achieve as the adversary has to actively perform downgrade attacks for each session.

Interactive fallback. Interactive fallbacks involve the user. If used in BEFS, when the FS-ciphersuites handshake fails, the client (e.g. web browser) presents an interrupting warning message and asks the user whether to proceed or not. If the user chooses to proceed, the client falls back from FS-ciphersuites to default ciphersuites and performs a second handshake. Otherwise, if the user chooses not to proceed, the client does not fall back, and aborts the TLS handshake. Interactive fallbacks provide security against active adversaries. Interactive fallbacks are similar to the widely-known self-signed certificate active warnings [1]. Active security warnings have been shown to be more effective than passive ones such as passive indicators that do not interrupt the user's task [1]. However, active security warnings have to be used with caution in order to not cause the habitation effect, where users ignore them because they see them too often [38]. Therefore, if the majority of servers that do not support FS-ciphersuites (i.e. those that require fallback) are network devices, interactive fallback can be acceptable, as these devices are normally visited infrequently by a limited number of users, such as the device's owner.

Signaled fallback. Signaled fallbacks involve the server. Therefore, if they are not incorporated in the protocol by design, they require modifications or updates to the server, e.g. a patch to the TLS implementation, to enable the server from interpreting the client's signal. In signaled fallbacks, the client sends a signal, i.e. a special value, to inform the server that the client has performed a fallback. The server aborts the handshake if it is not expecting a fallback, e.g. in BEFS case, if the server supports FS-ciphersuites. Signaled fallbacks provide security against active adversaries, if we assume authenticated messages. Signaled fallbacks have been proposed in the TLS fallback Signaling Cipher Suite Value (SCSV) [26]. It has been used to mitigate TLS version downgrade attacks, mainly the POODLE attack [27], and has been widely adopted as shown in [7]. In BEFS, our problem deals with misconfigured servers and less security-aware server administrators. Had they been security-aware, they would have configured their servers to select FS-ciphersuites. Therefore, in BEFS case, we do not consider signaled fallbacks as a solution that can be adopted quickly. Therefore, we do not include it in our analysis in the coming section.

BEFS Security Analysis We now analyse the security of BEFS against three adversarial models: passive network adversary, active network adversary, and our newly introduced discriminatory adversary.

Passive Network Adversary Passive adversaries can collect network traffic, but cannot interfere (e.g. modify, inject, replay, or drop) protocol messages. They may obtain access to the server's long-term private-key at some point in the future. Once the server's long-term private-key is compromised, a passive adversary who has been collecting non-FS network traffic can now decrypt it. BEFS aims to ensure the selection of FS-ciphersuites whenever the server supports FS-ciphersuites. In FS-ciphersuites, an ephemeral key is generated for each session, and this key is not encrypted with the server's long-term private-key. By selecting FS-ciphersuites, if the server's long-term private-key is compromised, the adversary cannot compromise past session keys. In TLS, the (EC)DHE key-exchange algorithms are provably secure against passive ad-

versaries [19]. Therefore, BEFS with all types of fallback mechanisms is secure against passive adversaries.

Active Network Adversary Unlike passive adversaries, active adversaries can interfere with protocol messages, e.g. by modifying, injecting, replaying, or dropping messages. Similar to the passive adversaries, they may obtain access to the server's long-term private-key at some point in the future, hence be able to decrypt non-FS-ciphersuite traffic. BEFS security against active adversaries can be analysed with the two fallback mechanisms explained earlier in section 4.2. First, in terms of BEFS with silent fallback, since the user of a BEFS-enabled client with silent fallback is not aware of the fallback, an active adversary can perform a downgrade attack by dropping the initial FS-ciphersuites handshake message to lead the client to fall back and perform a default handshake. Hence, misconfigured servers that select non-FS-ciphersuites but support FS-ciphersuite will not be guided, i.e. will select non-FS-ciphersuite, while with BEFS, they will be guided (forced) to select FS-ciphersuites instead. BEFS with silent fallback does not provide security against active adversaries. Second, we analyse BEFS with interactive fallback against an active adversary. This moves the security decision to the user. Users can be classified into two categories: security-aware users, who read the warning message and reject the fallback when they care about FS. The second category of users is less security-aware users, who will not do so. BEFS with interactive fallback and security-aware users is secure against active adversaries. The warning message content and the users' reactions to it are beyond the scope of this paper. BEFS with interactive fallbacks can find its application in special browser modes for sensitive communications, in the same vein of `Chrome`'s incognito and `Firefox` private modes, which are available for privacy-aware users.

Discriminatory Adversary The discriminatory adversary is located at the server and discriminates against its clients in terms of the security level it provides to them (FS-ciphersuite vs. non-FS-ciphersuite in our case). The discriminatory adversarial model is applicable to semi-trusted servers running protocols such as TLS, which gives the server the power of selecting some parameters that define the security level of a particular session, exemplified by the ciphersuite in our case, while the client has no means of verifying the server's actual capabilities, i.e. justifying the server's decision if it selects a non-preferred ciphersuite such as a non-FS-ciphersuite. This power can be abused by semi-trusted servers to discriminate against their users, for a powerful third-party's advantage. The discriminatory adversary can be compelled by, or collude with the third-party, such as government intelligence, to weaken the security of *some* connections, e.g. those coming from specific geographic locations. In our case, the discriminatory adversary denies the FS property to some users, whilst enabling it for others. The discriminatory adversary (server) can then provides its long-term private-key that is used for digital signatures and non-FS session keys (*pms*) encryption to the powerful third-party, after the key's expiration, when it is no longer used by the server. This allows the third-party to decrypt the data of those users who have been discriminated against, but not the data of other users who have been provided with strong security, i.e. FS-ciphersuite in our case. This adversarial model gives the semi-trusted server several advantages compared to giving every session key or the decrypted data itself to the third-party, which is impractical for servers to carry out, especially in the case of large-scale surveillance.

Another advantage to the semi-trusted server lies in the minimal liabilities (e.g. legal) in being directly involved in leaking their users' data, or in giving their private-key to the adversary during the key's lifetime. Such an adversarial model is not far from the export-grade cryptography law that was mandated until the late 90s, where software vendors, for example, were compelled to weaken the security of software exported to outside the United States (US), to enable US intelligence from breaking their security. Furthermore, leaked confidential documents by Edward Snowden suggests similar scenarios, where giant companies collude with government intelligence by introducing backdoors that are known to, and can be exploited by, those powerful adversaries (e.g. the "PRISM" program) [42].

Our discriminatory adversarial model is inspired by the "malicious-but-cautious" [35] and the Secretly Embedded Trapdoor with Universal Protection (SETUP) [44] adversarial models. The "malicious-but-cautious" model assumes a cloud service provider (server) can act maliciously but is cautious not to leave a verifiable trace of its malicious behaviour. However, it does not assume that the malicious server is willing to enable a third-party to access some users' data. On the other hand, the SETUP model assumes a cryptographic system (server in our case) can enable a third-party to secretly obtain secret information such as the private-key that decrypts the encrypted data from the system's encrypted output. Our discriminatory adversary weakens the security against some users for a third party's advantage, and is also cautious not to leave a verifiable trace of its malicious behavior, e.g. by selecting a supported but non-preferred ciphersuite (non-FS-ciphersuite), as it is still accepted by most clients for backward compatibility.

To better analyse BEFS against the discriminatory adversary (server), we further classify this adversary into two variants: weak discriminatory and strong discriminatory. In the weak variant, the adversary submits to the client's offer (ciphersuites in our case). That is, if the client offers strong choices exclusively, the weak discriminatory has no choice but to select from them, mainly to avoid detection. In the strong variant, the adversary refuses to select strong choices, which forces the client to fallback in order to connect to the server. BEFS with silent fallback is secure against the weak discriminatory adversaries. However, strong discriminatory adversaries require interactive fallback and security-aware users. In today's real-world settings, the weak variant is more realistic. However, the strong variant can be detected through BEFS and security-aware users. Note that this analysis of BEFS against a discriminatory adversary is independent of considerations about the communication channel. That is, if an active adversary is present in the communication channel, interactive fallback is required with both variants of the discriminatory adversary, in order for BEFS to meet its security goal.

Best Effort Forward Secrecy and Authenticated Encryption (BESAFE) Given the fact that more than 50% of the servers select FS+non-AE-ciphersuite after enforcing FS-ciphersuites and that between 16.14% to 25.79% of them support FS+AE-ciphersuites, as an extension to BEFS, we propose BESAFE which adds an additional step to enforce not only FS-ciphersuites, but also FS+AE-ciphersuites. This improvement adds an additional restriction: the client offers FS+AE-ciphersuites *exclusively* at the first handshake attempt. If it failed, the client falls back to BEFS: it tries FS-ciphersuites *exclusively*,

and if it failed, it falls back to default ciphersuites. The BESAFE mechanism guides servers towards FS+AE-ciphersuites. Similar to BEFS, BESAFE is secure against passive adversaries, or weak discriminatory adversaries with all types of fallbacks, and against active adversaries, or strong discriminatory adversaries with interactive fallback and security-aware users.

BEFS and BESAFE Performance We measure the latency that BEFS and its extension BESAFE incur into a TLS connection establishment with domains that do not support FS-ciphersuites, i.e. when more than one attempt is performed to complete a TLS handshake (otherwise, in BEFS, if the server supports FS-ciphersuites, and in BESAFE, if the server supports FS+AE-ciphersuites, there will be a single handshake as normal and no additional latency is incurred). To this end, we extract 5000 top domains that do not support FS-ciphersuites from our results. We implement a TLS client that supports `Chrome`'s pre-TLS 1.3 default ciphersuites using `Python 3.6.5` and utilising `OpenSSL 1.1.0g`. We disable TLS certificate validation and session tickets (resumption), and enable the SNI. Our client performs three consecutively handshakes for each domain: Default, BEFS-enabled, and BESAFE-enabled handshakes. We run the client on a machine equipped with a 3.2 GHz `Intel Core i5` processor, 8 GB of RAM, and a 1000 Mbps wired Ethernet card that has a public IPv4 address at the University of Oxford. We measure the time to complete a TLS handshake in a socket connection in milliseconds using the `process_time()`, a process-wide timer in python's `time` module. We count the domains that triggered BEFS and BESAFE to resort to default ciphersuites (i.e. do not support FS-ciphersuites), and also responded to the default TLS handshake. Then we extract the maximum, minimum and average time they take for each of the three handshake types. There are 4501 domains that do not support FS-ciphersuites and responded to the three types of handshakes we examine. The results based on these responses are summarised in Table 3. We can also infer the latency that BESAFE incurs into a connection to a server that does not support FS+AE-ciphersuites but supports FS+non-AE-ciphersuites from the BEFS latency (since both require two attempts).

Table 3: The BEFS and BESAFE mechanisms latency in ms compared to the default one when connecting to servers that do not support FS-ciphersuites.

TLS Client	Max.	Min.	Avg.
`Default`	4.10	0.64	1.69
`BEFS-Enabled`	5.34	1.79	3.47
`BESAFE-Enable`	8.60	3.27	5.19

Improved Performance Through Parallel Attempts As shown in the previous section, BEFS introduces a latency on the default TLS connection establishment, but only if the server does not support FS-ciphersuites. To minimise this latency, the client can implement BEFS attempts in parallel instead of consecutive. That is, the client sends two `CH`s one with default ciphersuites and the second with FS-ciphersuites, in parallel to the server. For each TLS session establishment, the client waits for all the `CH` attempts' responses to return. If there is a valid response to the FS-ciphersuites attempt, the client proceeds with the FS-ciphersuites response. Otherwise, if the FS-ciphersuites attempt has failed, the client proceeds with the default ciphersuite response. The same applies to BESAFE but the client sends three handshakes and first checks the FS+AE-

ciphersuites, then the FS-ciphersuites attempts' response, before deciding to proceed with default ciphersuites.

5 Related work

Lee et al. [24] scanned around 19,000 TLS servers based on top domains lists. They evaluate the cryptographic strengths in TLS servers including the supported key-exchange algorithms. In [17], Holz et al. provide statistics for the most popular selected ciphersuite by TLS servers. More recently, Kotzias et al. [21] examined the impact of high-profile attacks on TLS deployment, and Calzavara et al. [9] analysed the `Alexa` top 10,000 websites against known HTTPS vulnerabilities. All the aforementioned studies do not analyse the selected versus supported key-exchange algorithms as we do. In [18], Huan et al. provide an experimental study on TLS FS deployment on 473,802 of `Alexa`'s top domains using an enumeration-based method. Our study analyses larger and more diverse datasets than that in [18], using a novel heuristic approach. Additionally, our study provides a recent view on FS adoption over the one in [18], which dates back to 2014. Apart from measurement studies, several studies show that RSA long-term private-keys can be compromised either due to advances in computing power, deployment, or implementation flaws. Kleinjung et al. [20] and Cavallar et al. [10] show that 786-bit and 512-bit RSA keys can be factored using powerful machines. Heninger et al. [16] conducted a measurement study, which is replicated by Alashwali [2], that shows that factorable RSA keys are widespread in network devices on the Internet, due to low entropy during prime generation. The Heartlbleed bug [39] in the `OpenSSL` TLS library shows that implementation bugs can cause private-key compromise.

6 Conclusions

In this paper, we analysed the state of FS on over 10 million servers on the Internet. Using a modern TLS client handshake algorithms, our results show 5.37% of top domains, 7.51% of random domains, and 26.16% of random IPs *do not select* FS key-exchange algorithms. Surprisingly, we found that 39.20% of the top domains, 24.40% of the random domains, and 14.46% of the random IPs that *do not select* FS, *do support* FS. We then discussed possible paths towards FS. We showed that a best effort approach can add a value over the "all or nothing" approach and can increase FS or FS and AE adoption in misconfigured servers.

Acknowledgment

We thank the `Censys` team [13], the CS's IT and OxCERT teams at the University of Oxford, and the `tls-scan` developer, Binu Ramakrishnan, for technical support. Pawel's work was supported by the SUTD SRG ISTD 2017 128 grant.

References

1. Akhawe, D., Felt, A.P.: Alice in Warningland: A Large-Scale Field Study of Browser Security Warning Effectiveness. In: Proceedings of USENIX Security Symposium (2013)
2. Alashwali, E.S.: Cryptographic Vulnerabilities in Real-Life Web Servers. In: Proceedings of Int. Conference on Communications and Information Technology (ICCIT). pp. 6–11 (2013)

3. Alashwali, E.S., Rasmussen, K.: What's in a Downgrade? A Taxonomy of Downgrade Attacks in the TLS Protocol and Application Protocols Using TLS. In: Proceedings of Applications and Techniques in Cyber Security (ATCS) (2018)
4. Alashwali, E.S., Szalachowski, P., Martin, A.: Does "www." Mean Better Transport Layer Security? In: Proceedings of Availability, Reliability and Security (ARES) (2019)
5. Alexa Internet, Inc.: Alexa Top Sites (2018), http://s3.amazonaws.com/alexa-static/top-1m.csv.zip, accessed Aug. 22, 2018
6. AlFardan, N.J., Paterson, K.G.: Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. In: Proceedings of Security and Privacy (SP). pp. 526–540 (2013)
7. Amann, J., Gasser, O., Scheitle, Q., Brent, L., Carle, G., Holz, R.: Mission Accomplished?: HTTPS Security After Diginotar. In: Proceedings of Internet Measurement Conference (IMC). pp. 325–340 (2017)
8. Barnes, R., Thomson, M., Pironti, A., Langley, A.: Deprecating Secure Sockets Layer Version 3.0 (2015), https://tools.ietf.org/html/rfc7568, accessed Sept. 30, 2018
9. Calzavara, S., Focardi, R., Nemec, M., Rabitti, A., Squarcina, M.: Postcards from the Post-HTTP World: Amplification of HTTPS Vulnerabilities in the Web Ecosystem. In: Proceedings of Security and Privacy (SP) (2019)
10. Cavallar, S., Dodson, B., Lenstra, A.K., Lioen, W., Montgomery, P.L., Murphy, B., te Riele, H., Aardal, K., Gilchrist, J., Guillerm, G., Leyland, P., Marchand, J., Morain, F., Muffett, A., Putnam, C., Craig, Zimmermann, P.: Factorization of a 512-Bit RSA Modulus. In: Proceedings of Advances in Cryptology (EUROCRYPT). pp. 1–18 (2000)
11. Diffie, W., Hellman, M.: New Directions in Cryptography. IEEE Transactions on Information Theory **22**(6), 644–654 (1976)
12. Dukhovni, V.: Opportunistic Security: Some Protection Most of the Time (2014), https://tools.ietf.org/html/rfc7435.html, accessed Oct. 1, 2018
13. Durumeric, Z., Adrian, D., Mirian, A., Bailey, M., Halderman, J.A.: A Search Engine Backed by Internet-Wide Scanning. In: Proceedings of Computer and Communications Security (CCS). pp. 542–553 (2015)
14. Eastlake 3rd, D.: Transport Layer Security (TLS) Extensions: Extension Definitions, https://tools.ietf.org/html/rfc6066#page-6, accessed Jun. 19, 2019
15. FIPS: Advanced Encryption Standard (AES) (2001), https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf, accessed Sept. 30, 2018
16. Heninger, N., Durumeric, Z., Wustrow, E., Halderman, J.A.: Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices. In: Proceedings of USENIX Security Symposium (2012)
17. Holz, R., Braun, L., Kammenhuber, N., Carle, G.: The SSL Landscape: a Thorough Analysis of the X. 509 PKI Using Active and Passive Measurements. In: Proceedings of Internet Measurement Conference (IMC). pp. 427–444 (2011)
18. Huang, L.S., Adhikarla, S., Boneh, D., Jackson, C.: An Experimental Study of TLS Forward Secrecy Deployments. IEEE Internet Computing **18**(6), 43–51 (2014)
19. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the Security of TLS-DHE in the Standard Model. In: Proceedings of Advances in Cryptology (CRYPTO). pp. 273–293 (2012)
20. Kleinjung, T., Aoki, K., Franke, J., Lenstra, A.K., Thomé, E., Bos, J.W., Gaudry, P., Kruppa, A., Montgomery, P.L., Osvik, D.A., te Riele, H., Timofeev, A., Zimmermann, P.: Factorization of a 768-bit RSA Modulus. In: Proceedings of Advances in Cryptology (CRYPTO). pp. 333–350 (2010)
21. Kotzias, P., Razaghpanah, A., Amann, J., Paterson, K.G., Vallina-Rodriguez, N., Caballero, J.: Coming of Age: A Longitudinal Study of TLS Deployment. In: Proceedings of Internet Measurement Conference (IMC). pp. 415–428 (2018)
22. Kurkowski, J.: tldextract (2017), https://github.com/john-kurkowski/tldextract, accessed Oct. 30, 2018

23. Laurie, B., Langley, A., Kasper, E.: Certificate Transparency (2013), accessed Feb. 25, 2019
24. Lee, H.K., Malkin, T., Nahum, E.: Cryptographic Strength of SSL/TLS Servers: Current and Recent Practices. In: Proceedings of Internet Measurement Conference (IMC). pp. 83–92 (2007)
25. Menezes, A.J., Van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC press (1996)
26. Moeller, B., Langley, A.: TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks (2014), https://tools.ietf.org/html/draft-ietf-tls-downgrade-scsv-00, accessed Oct. 1, 2018
27. Möller, B., Duong, T., Kotowicz, K.: This POODLE Bites: Exploiting the SSL 3.0 Fallback (2014), https://www.openssl.org/~bodo/ssl-poodle.pdf, accessed Jul. 6, 2018
28. Nir, Y., Josefsson, S., Pegourie-Gonnard, M.: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier (2018), https://tools.ietf.org/html/rfc8422, accessed Jun. 21, 2019
29. Partridge, C., Allman, M.: Ethical Considerations in Network Measurement Papers. Communications of the ACM **59**(10), 58–64 (2016)
30. Popov, A.: Prohibiting RC4 Cipher Suites (2015), https://tools.ietf.org/html/rfc7465, accessed Sept. 30, 2018
31. Qualys Inc.: SSL Labs (2018), https://www.ssllabs.com/ssl-pulse/, accessed Apr. 10, 2019
32. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2 (2008), https://www.ietf.org/rfc/rfc5246.txt, accessed Jul. 6, 2018
33. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3 draft-ietf-tls-tls13-28 (2018), https://tools.ietf.org/html/draft-ietf-tls-tls13-28, accessed Jul. 6, 2018
34. Rivest, R.L., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM **21**(2), 120–126
35. Ryan, M.D.: Enhanced Certificate Transparency and End-to-End Encrypted Mail. In: Proceedings of Network and Distributed System (NDSS) (2018)
36. Salowey, J., Choudhury, A., McGrew, D.: AES Galois Counter Mode (GCM) Cipher Suites for TLS (2008), https://tools.ietf.org/html/rfc5288#page-3, accessed Nov. 12, 2018
37. Samarasinghe, N., Mannan, M.: Short Paper: TLS Ecosystems in Networked Devices vs. Web Servers. In: Proceedings of Financial Cryptography and Data Security (FC). pp. 533–541 (2017)
38. Sunshine, J., Egelman, S., Almuhimedi, H., Atri, N., Cranor, L.F.: Crying Wolf: An Empirical Study of SSL Warning Effectiveness. In: Proceedings of USENIX Security Symposium. pp. 399–416 (2009)
39. Synopsys Inc.: The Heartbleed Bug (2014), http://heartbleed.com, accessed Sept. 17, 2018
40. Vaudenay, S.: Security Flaws Induced by CBC Padding-Applications to SSL, IPSEC, WTLS.... In: Proceedings of Theory and Applications of Cryptographic Techniques (EUROCRYPT). pp. 534–546 (2002)
41. W3Schools: Browser Statistics (2019), https://www.w3schools.com/browsers, accessed Feb. 27, 2019
42. Wikipedia: PRISM (Surveillance Program) (2018), https://en.wikipedia.org/wiki/PRISM_(surveillance_program), accessed Oct. 3, 2018
43. Yahoo Inc.: tls-scan (2016), https://github.com/prbinu/tls-scan, accessed Sept. 8, 2018
44. Young, A., Yung, M.: The Dark Side of Black-Box Cryptography or: Should We Trust Capstone? In: Proceedings of Advances in Cryptology (CRYPTO). pp. 89–103 (1996)