PhD First Year Report: Information Leakage on Encrypted Sensor Network Traffic

Yan Yan

April 29, 2015

Contents

Chapter 1

Introduction

1.1 Cryptography and Side Channel Attacks

Cryptography derived from solving the problem of transmitting secret information through an insecure channel. The general principle in modern cryptography is that the encryption and decryption algorithms are assumed to be made public and the secrecy of messages is solely relied on the secrecy of the key.

Under this assumption, cryptographic algorithms are then designed in various ways. Some of them are derived from hard mathematical problems such as Discrete Logarithm and Elliptic Curves. Provable Security provides mathematical proofs by modelling the algorithms in an abstracted mathematical world and reduces them to hard problems; the secrecy of message is then guaranteed as long as the underlying mathematical problems are not solved.

However, when implemented in real world things can get out of control. Many factors considered difficult to be modelled mathematically can sometimes be exploited to breach the security. For example, [1] describes a method called Differential Power Analysis which recovers the secret key using power traces measured during encryption; [2] and [3] shows that the timing information can also reveal the secret key.

1.2 Network Traffic

The conceptional structure of today's network is described in [4]. Generally speaking, networks are formed by protocols which are standard agreements implemented by nodes connected to the networks. Protocols are categorised as a stack of layer with the lower ones handle the fussy transportation problems and the higher ones the more sophisticated functional problems.

Packet (or frame) is the unit of structured data transmitted through network at each layer. A typical format of packet is a protocol header followed by its payload. The header constitutes of several fields which contain some meta data of the packet while the payload is the actual data being transmitted.

	Layer	Example Protocols
High	Application Layer	HTTP / SMTP / NTP
	Secure Layer	TLS / SSL / DTLS
	Transport Layer	TCP / UDP / SCTP
	Network Layer	IPv4 / IPv6 / 6lowPAN
	Data Link Layer	IEEE 802.3 / IEEE 802.11
Low	Physical Layer	Bluetooth / DSL

Table 1.1: A Simplified 6 Layer Model

Packet of protocols are organised recursively, i.e. headers of upper protocols is embedded into the payload of lower protocols. When a packet (or frame) arrives a node on the network, the headers are decapsulated sequentially from the lowest layer to the highest. Conceptional speaking, the content of a payload should be transparent to the protocols beneath. For instance, a router may only decapsulate a packet no higher than IP layer and has no concern about everything above.

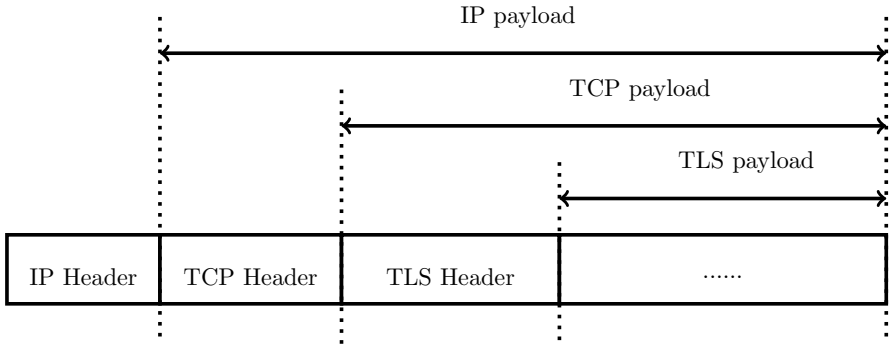

Figure 1.1: An example of headers and payloads

It might be difficult in real world to prevent a third party from seeing a packet. For example, a packet from your laptop cannot reach Google server without hopping between routers and hence any corrupted router may breach the data; the enriching application of mobile devices made it even more difficult as the radio signals from one's smart phone can be captured easily in a cafe.

1.3 Side Channel Information in Packets

Theoretically speaking, encryption can be applied at any layer of a network. Lower the layer it is, more information is protected. However, the intense requirement of efficiency and bandwidth have limited the usage of cryptography, like IPSec. A trade-off between security and efficiency must be made and encryption at higher level protocol has eventuall adopted as a widely accepted

solution. As a result, TLS/SSL has became the most used security protocol on Internet today.

The drawback, as mentioned above, is that all headers in the lower layers will be transmitted in plaintext. Figure 1.2 is an example of a captured DTLS packet.

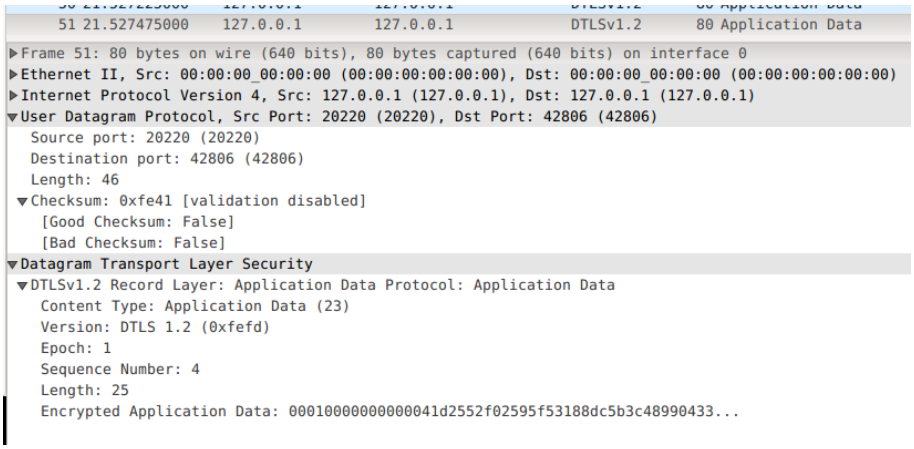

Figure 1.2: A Wireshark Screenshot

As we can see in Figure 1.2, although the sniffer is not able to decrypt the packet, all headers until DTLS are indeed accessible.

1.4 Motivation and Challenges

The open nature of the development Internet of Things (IoT) demands more security and privacy measure. Unlike the stable and already-widely-deployed Internet, the structure of IoT network is still under a developing process; therefore it might be worthwhile to take into concern the security from an early stage.

One challenge in this project is the fundamental nature difference between network traffic and other physical side channel information. Many side channel attacks are aimed to recover secret key materials, such as DPA in [1], or Cache-timing Attack in [2] and [3].

Although it is not certain at this stage that whether there is a link between the value of cryptographic key chosen and packet headers, but intuitively they are more likely to be independent. The reason is that the cryptography primitives chosen should only affect the value of ciphertext which constitutes the payload; hence is expected to be transparent to the lower layer protocol headers. However, the timing information might still be exploited to recover the key material.

Different protocol suite result into different headers and hence different information. As a beginning for this project, we choose to study the traffic gen-

erated by a DTLS implementation called tinyDTLS[5]. tinyDTLS is designed to be suitable for embedded systems and is one of the IoT candidates in the future.

Chapter 2

Literature Review

2.1 Academic Publications

[6] demonstrated that this kind of leakage indeed exists in some web applications. Firstly, different web page has different page sizes; thus the size can be exploited as a fingerprint to indicate which page the victim accessed. Secondly, on some pages where part of its content is generated dynamically, such as suggestions in a search box, the size of the dynamic content can again be used as a fingerprint and therefore reveal the victim's input which is supposed to be kept secret.

[7] focuses on detecting leakage points of input in web applications. In this paper, a web applications is modelled as a finite state machine. Observable packet information, such as length and flags in a TCP header, are represented as a vector. A trace is then defined as a sequence of packet information. The leakage points are then measured by estimating the mutual information of the input and each point in the trace. As an extension to mutual information analysis, [8] and [9] provide methods to improve accuracy and performance of the test.

[6] and [7] together lay a foundation to this project. [6] focuses more on the fact that information leakage through encrypted network traffic actually exists but the attacks it describes are relatively web-site specific, although this could be a feature in this subject. The mutual information test proposed in [7] is a powerful tool to pinpoint the leakage points but it also suffers from a performance issue as it requires a huge amount of computation.

[10] and [11] described two actual attacks using packet length. [10] uses packet length to fingerprint pages which contains potentially privacy information. [11] attacks search keyword by profiling the size of response for each character typed in the search box, then matches the victim's traffic in a suffix tree built for the dictionary by a stochastic algorithm.

Packet length might be the most juicy target in packet analysis attacks. These are particularly efficient and requires only passive observation and not much effective countermeasures are proposed. However, the effectiveness of

these attacks might be affected by divergence of packet lengths which can be seen common as a result of ads or other kinds of dynamic contents in web pages. One potential problem in [11] is that the profiling step takes a long time due to the response time of search engine and the dictionary chosen also has a great impact on the efficiency and accuracy of attack. It might be worthy pointing out that most theoretical cryptographical research studies only secrecy of plaintexts with same length which is not the case in real world. I personally think that these attacks exploited a blind spot in theoretic study and have displayed the gap between theoretical world and practical world. Packet length is also the most studied side channel information in our experiment at this stage. (See Chapter 3)

Compression ratio obtained by toggling the compress flag in some protocol can also be exploited to recover plaintext. The general idea of this kind of attack is described in [12]. This attack requires the adversary being able to inject his guesses of plaintext into the uncompressed plaintext. The compression ratio will be relatively higher when the guess matches part of the plaintext and lower if it does not. [13] and [14] gave implementations against TLS and HTTPS in real scenarios respectively. [15] proposed some countermeasures by disabling compression or inducing some length hiding mechanism.

At a first glance, the partial plaintext control assumption in [12] seems to be weird, but [14] has demonstrated that such circumstances can be practically achieved by a phishing link. Further more, some crucial information can be attacked such as the session cookie which can then be used to hijack a https session, like on-line banking. As an impact of these attacks, the latest TLS standard has disabled the usage of compression. Nevertheless, when it comes to constrained environment where bandwidth becomes precious, a solution better than totally forbidden is desired. However, there is no compression in our current set up(see Chapter 3). This kind of attack is not considered at this stage.

[16] exploits the padding scheme of CBC mode[1] to recover the plaintext. This attack is based on the fact that some SSL implementation returns different error messages on padding error and MAC failure. Since the padding verification is done after decryption but before MAC verification, the adversary can send to the server a partially modified ciphertext and recover the plaintext by different error messages. Although one countermeasure against this attack is to unify the error messages, [17] states that it is still possible to distinguish these errors by measuring the time difference induced by whether a MAC verification is performed.

The countermeasure against the attack in [16] has been adopted in many recent implementations by unifying the error message. The attack described [17] relies on the heartbeat being turned on and is very sensitive to network latency. Above that as stated in the paper that adding a idle time before respond can effectively prevent this attack. For a sensor network, the time resolution

[1] `http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_Block_Chaining_.28CBC.29`

is expected to be higher due to the low processing power of devices. It is not clear at this stage whether this will raise the risk of this attack. However, the cipher suite adopted by our current experimental environment does not have any padding at all(see Chapter 3); hence this kind of attack is not being considered at this stage.

2.2 Standard Specifications

Another part of the related literatures are the RFC specifications[2]. As of this project, we tend to put our focus on the protocols from Network Layer to Secure Layer in Table 1.1 as a consideration of generality. Therefore, the following documents can be regarded as main references of protocol specifications:

- RFC 791[18] is the IPv4 specification. IPv4 is the most widely deployed protocol today. Even though its successor IPv6 has been proposed for more than a decade, the replacing process is taking place very slowly. It is expected that IPv4 will remain its dominance over the next few years.
- RFC 2460[19] is the IPv6 specification. IPv6 is the successor of IPv4. It was published in 1998 due to the exhaustion of IPv4 address. Although kernels today usually support both IPv4 and IPv6, many applications are still based on IPv4. However, as a "next generation" network, many IoT manufacturers also have the trend of adopting IPv6 as their standard.
- RFC 6282[20] is the specification for 6lowPAN. 6lowPAN is a compression header format for IPv6. It is designed for applications over constrained environments such as sensor networks where the resources, like bandwidth, are very limited. Comparing to a standard IPv6 header, a 6lowPAN header omits some fields by setting them to a default value suitable for constrained environments and hence saved some bandwidth as well as processing time.
- RFC 6347[21] is the DTLS specification. DTLS is the counterpart of TLS over UDP. Since DTLS relies on UDP instead of TCP therefore it is considered to be more lightweight but unreliable. Even though UDP is designed to be connectionless in order to reduce its overhead, DTLS additionally (comparing to TLS) implemented some connection-oriented feature such as sequence to remain functional.
- RFC 7252[22] is the specification document for Constrained Application Protocol, CoAP. CoAP is a general application protocol for constrained environment. It has a similar design to HTTP which has a Request-Respond model and supports methods including GET, POST, PUT and DELETE. CoAP is designed to be able to map to HTTP for interoperability purpose.

[2] `http://www.ietf.org/rfc.html`

Chapter 3

Progress To Date

As a beginning of this project, our first step is to demonstrate that information leakage similar to those described in [6] can be found when the underlying protocol is switched DTLS. The reason is that DTLS is more suitable to sensor networks comparing to TLS due to is relatively lightweight-ness.

The basic idea is to build some toy applications which model typical sensor network traffic generated through DTLS. The information leakage of toy applications are intentionally crafted to emphasise their existence.

Even though both OpenSSL and GnuSSL have DTLS implemented with general features, we setted our experimental the less featured tinyDTLS[5] due to its lightweight-ness, which is more suitable to sensor networks. However, the drawback is that only one cipher-suite, TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8[23], is available for the current version of tinyDTLS. This implies that there should be no padding scheme adopted and hence the length of plaintext and ciphertext are expected to have a linear relationship. Our experiments supported this conjecture such that:

$$l_D = 17 + l \tag{3.1}$$

where l is the length of plaintext and l_D is the value in DTLS length field. According to the specifications the additional bytes is supposed to be purely a resulted of the appending MAC even though 17 bytes is a value unlikely to be. This problem is still under investigating.

All experiments are done with only two processes, a server and a client referred as SERVER and CLIENT, on a same linux host connected through local-link. The protocol suite we adopted is [IPv4 or IPv6] + UDP + DTLS. The modelled adversary is simply a passive eavesdropper.

So far we have built up two toy applications, **Odd or Even** and **Leaky Coffee**, which will be explained in the next sessions alongside with corresponding traffic analysis.

3.1 Odd or Even

Odd or Even is a simple toy application designed to demonstrate the fundamental idea of encrypted traffic analysis.

3.1.1 Application Description

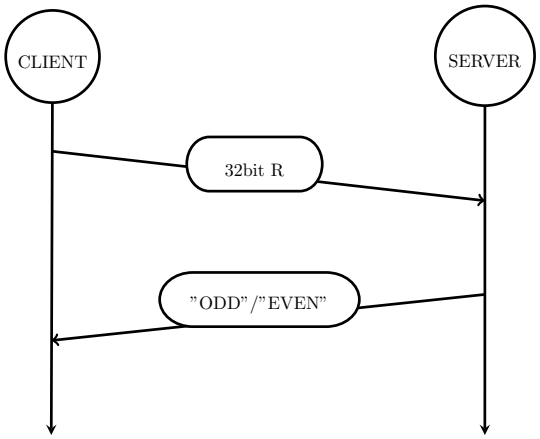

Figure 3.1: Description of an Odd-or-Even session

CLIENT randomly generates a 32-bit unsigned integer R and sends it to SERVER in binary. SERVER replies with a string "ODD" or "EVEN" according to the value of the 32-bit R(Figure 3.1).

3.1.2 Analysis

We run the application for multiple times and collected the packets it generated. As we have expected, "ODD" packets are 1 byte shorter than "EVEN" packets which implies that an eavesdropping adversary can learn what has been sent from SERVER to CLIENT simply by looking at the packets length. However, no obvious leakage has been found in other fields of the packets.

3.2 Leaky Coffee

3.2.1 Application Description

Leaky Coffee simulates a more complicated scenario where the CLIENT sends a coffee order (in string) to SERVER. SERVER echoes the order appended by some flavour (in string). CLIENT compares the amount of given flavour to an internally generated random requirement and asks SERVER again for more additive if it is insufficient. (Figure 3.2)

There are two examples described in Example 3.2.1 and Example 3.2.2.

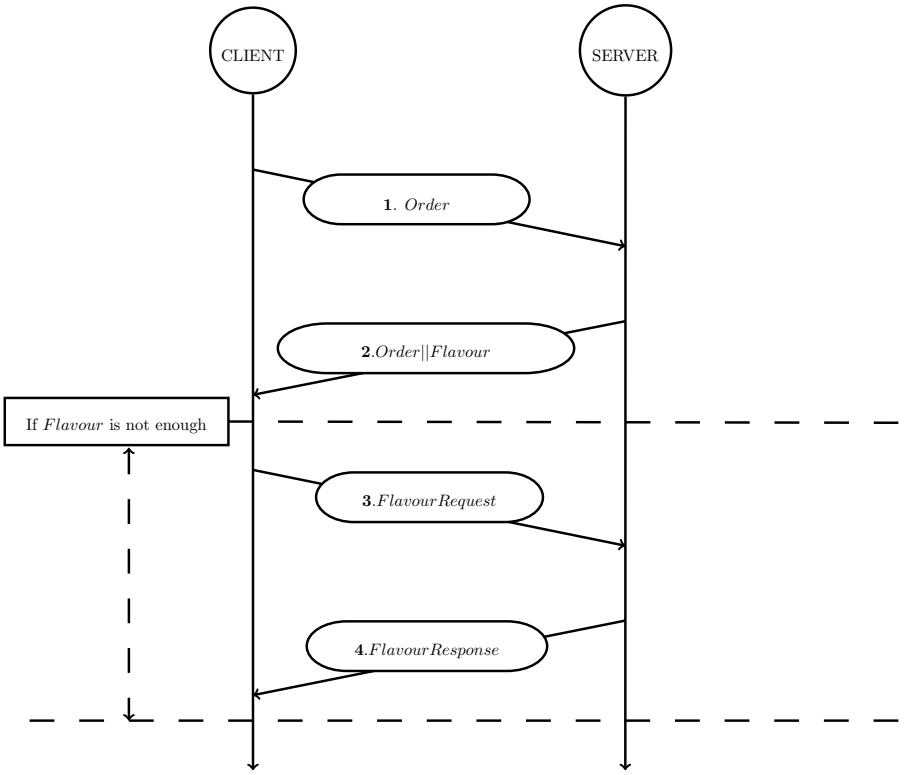

Figure 3.2: Description of a Leaky-Coffee session

Syntax

Definition 3.2.1. *COFFEE* is a set of strings defined as:
$COFFEE = \{\text{"AMERICANO"}, \text{"CAPPUCCINO"}, \text{"ESPRESSO"}, \text{"MOCHA"}\}$

Definition 3.2.2. Let '*' represents SUGAR and '@' represents MILK respectively, we denote n_* and $n_@$ as the number of their appearances in a string. We also call n_* and $n_@$ the degree of SUGAR and MILK respectively.

Definition 3.2.3. We define another set of string *ADDITIVE* as:
$ADDITIVE = \{\{SUGAR, MILK\}^{0-6} | 0 \leq n_* \leq 3, 0 \leq n_@ \leq 3\}$.

In another word, an instance of *ADDITIVE* contains no more than 3 SUGAR and MILK.

Leaky-Coffee Session

A Leaky-Coffee session can be described as in **??**:

1 As an initiation of a session, CLIENT randomly picks a string $Order \in COFFEE$ and sends it to SERVER.

2 Upon receiving an *Order*, SERVER replies with a string $\{Order||Flavour\}$ where $Flavour \in ADDITIVE$ and $||$ represents concatenation. If $Order =$ "ESPRESSO" then the degrees of both SUGAR and MILK of *Flavour* are set to 0.

3 CLIENT randomly generates a SUGAR requirement $r_* \in [0, 3]$ and a MILK requirement $r_@ \in [0, 3]$. Then it scans the reply from **2** and computes its degrees of SUGAR and MILK. If any of the degrees does not met the requirements, i.e. $n_* < r_*$ and/or $n_@ < r_@$, then CLIENT sends a $FlavourRequest = \{"FLAVOUR"||\{SUGAR\}^{\max(r_*-n_*,0)}||MILK^{\max(r_@-n_@,0)}\}$.

4 If SERVER receives a *FlavourRequest*, it echoes back *FlavourRequest* as its *FlavourResponse*, i.e. $FlvaourResponse = FlavourRequest$.

Note that the *FlavourRequest* and *FlavourResponse* packets are probabilistic in a Leaky-Coffee Session.

Example 3.2.1. An example with *FlavourRequest* and *FlavourResponse*(Figure 3.3):

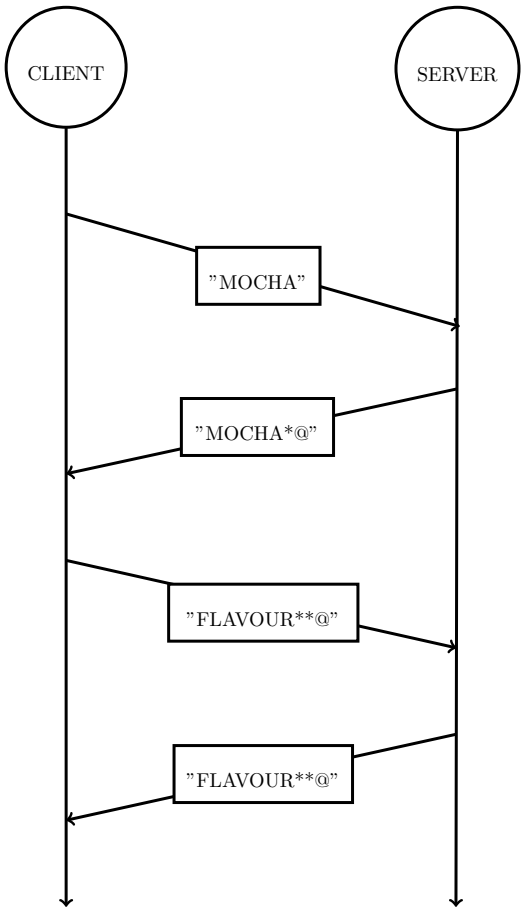

Figure 3.3: Example: A Leaky-Coffee session with *FlavourRequest* and *FlavourResponse*

In this example, CLIENT first sends an *Order* "MOCHA". SERVER then replies with "MOCHA*@" which implies both the SUGAR degree and MILK degree are 1. CLIENT randomly generates a SUGAR requirement 3 and MILK requirement 2 and then sends a *FlavourRequest* to request the shorted SUGAR and MILK. SERVER finally response with the requested ADDITIVE.

Example 3.2.2. Another example without *FlavourRequest* and *FlavourResponse*(Figure 3.4):

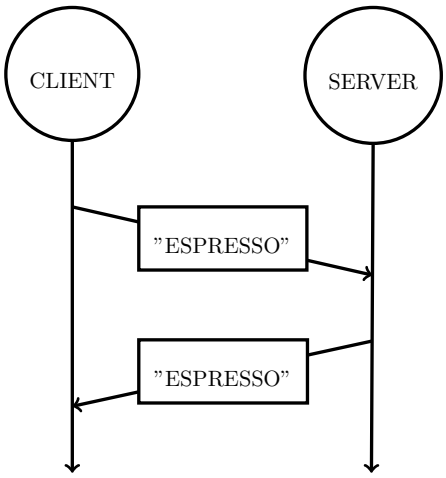

Figure 3.4: Example: A Leaky-Coffee session without $FlavourRequest$ and $FlavourResponse$

This example demonstrates a session initiated with "ESPRESSO" where no ADDITIVE will be added in the reply.

Some Implementation Details

SERVER listens to a fixed port (20220) while CLIENT assigns an ephemeral port during each run, i.e. CLIENT's port is selected at the beginning of each run and remains constant during the life time of that instance.

In this experimental implementation, all random values are generated by the Linux kernel random number generator(/dev/urandom); thus assumed to be uniformly distributed.

After each session, CLIENT will be putted into sleep for a random period from 5 to 15 seconds.

We used localloop as our network interface in our experiment; thus packet loss is not considered. DTLS does implement retransmission at some level, but since the sequence number in DTLS header does not change in the retransmitted packet so it is still seemingly possible to reconstruct the equivalent packet stream without any packet loss. Even though the reconstructed stream will preserve all information in each header but the accurate time stamps will be difficult to recover.

3.3 Analysis of Leaky Coffee

3.3.1 Session Detection

It is obvious that whenever there is a being packet transmitted then there is a session taking place.

3.3.2 Segmenting A Session from Continuous Traffic

Given the implementation, we can isolate a session from the packet stream by analysing their time stamp. This is achieved by using a threshold value and then compare it with the interval of two packets. If the interval is greater than the threshold then we can guess these packets belong to different sessions.

Algorithm 1: IsSameSession

Input: threshold θ, time stamps of two continuous packets t_1, t_2
Output: TRUE if the packets are of the same session, otherwise FALSE

```
1 if θ > |t2 − t1| then
2 |  return TRUE;
3 else
4 |  return FALSE
5 end
```

In our implementation, a typical guess .By continuously applying Algorithm 1 on the duplex packet stream, we can easily isolate different sessions.

3.3.3 Determine Packets in a Session

Once a session is segmented from the continuous traffic, it this not difficult to identify the type of each packet in **Leaky Coffee** as there can only be two types of session:

1. **Session with 4 packets**. This type of session can be identified with 4 packets presented. Further more, those packets can be identified sequentially as: $< Order, Order||Flavour, FlavourRequest, FlavourResponse >$ respectively as well.

2. **Session without *FlavourRequest* and *FlavourResponse***. 2 packets sessions can be identified as this type of session. Those packets can then be identified as $< Order, Order||Flavour >$ accordingly.

3.3.4 Guessing Plaintext in One Packet through Length

In this implementation, assume we have the pre-knowledge that each $Order \in COFFEE$ picked by CLIENT has an uniform distribution. Further from, the degree of SUGAR and MILK also have uniform distributions over $0, 1, 2, 3$. Given these distributions, it is some how possible to make guesses of the plaintext in the packets by the length given in DTLS header, or UDP header, without trying to break the encryption primitives.

We denote the value of DTLS Length field as l_D and the actual application data length as l. Our experiment shows that:

under both IPv4 and IPv6.

Definition 3.3.1. For a specific packet in a session, let $\mathbb{X}$ be the set of plaintext and $\mathbb{Y}$ be the set of its corresponding content length.

We model the plaintext and their corresponding content length (in bytes) as a channel:

$$W(y|x), x \in \mathbb{X}, y \in \mathbb{Y}.$$

And then the inverse of this channel $W^{-1}(x|y)$ can be viewed as the leakage channel of $\mathbb{Y}$.

The general idea is that with such leakage channel, an adversary can then "decode" the plaintext using this leakage channel.

In this context, $\mathbb{X}$ is the set of packet content and $\mathbb{Y}$ the set of content length l.

Example 3.3.1. We begin with a simple example: *Order*.

For *Order* packets, we have:

$W(y\|x)$	5	8	9	P
"AMERICANO"			1	1/4
"CAPPUCINO"			1	1/4
"MOCHA"	1			1/4
"ESPRESSO"		1		1/4

Table 3.1: Content-Length Channel and the probabilities of *Order*

In this implementation, CLIENT randomly picks *Order* from $COFFEE$; therefore the probability for every value is 1/4. Since neither DLTS nor the application induces any randomness to the content length therefore it will always be a deterministic value.

Given W and the probability of *Order*, it can then compute the joint distribution of $(Order, l)$ by:

$$(\widehat{WP})(x, y) = P(x)W(y|x) \tag{3.2}$$

$\widehat{WP}$	P
("AMERICANO",9)	1/4
("CAPPUCINO",9)	1/4
("MOCHA",5)	1/4
("ESPRESSO",8)	1/4

Table 3.2: Joint distribution of $(Order, l)$

Then follows the marginal distribution of content length:

$$P(Y = y) = \sum_{x \in \mathbb{X}} \widehat{W}P(x, y) \tag{3.3}$$

y	P
5	1/4
8	1/4
9	1/2

Table 3.3: Marginal distribution of l

Finally we can construct the leakage channel using Bayes' theorem:

$$P(x|y) = \frac{P(x)P(y|x)}{P(y)} \tag{3.4}$$

$W^{-1}(x\|y)$	"AMERICANO"	"CAPPUCINO"	"ESPRESSO"	"MOCHA"
5				1
8			1	
9	1/2	1/2		

Table 3.4: Leakage channel of Length - *Order*

The same strategy can also be applied on the second packet: $Order||Flavour$.

Example 3.3.2. The first step is to compute the Content-Length channel. Analysis on $Order||Flavour$ packet is more complicated as it has a larger entropy.

We omit the sequence of SUGAR and MILK to simplify the problem. We also simplify our notation by denoting D_1 as the degree of SUGAR and D_2 the degree of MILK. Then any $Flavour$ can be represented as (D_1, D_2), e.g. $(2, 1)$ represents any $Flavour$ that has a degree of SUGAR 2 and degree of MILK 1.

The same strategy can be applied directly on this example as well. However, the space of this channel is much more complicated in this case which are 4

It is sometimes possible to simplify the problem by breaking the Plaintext-Length channel into several sub-channels, namely *Order* channel $W_0(y \in l|x \in COFFEE)$, SUGAR channel $W_1(y \in l|x \in D_1)$ and MILK channel $W_2(y \in l|x \in D_2)$ in this application. These sub-channels requires less computation and we will show how to reconstruct the Plaintext-Length channel using these sub-channels later in this section.

Obviously that W_0 is identical to Table 3.1 as the *Order* part in $Order||Flavour$ is simply an echo of the first *Order* packet.

W_2 and W_3 are actually identical:

$W_1(x\|y)$	0	1	2	3	P
0	1				1/4
1		1			1/4
2			1		1/4
3				1	1/4

$W_2(x\|y)$	0	1	2	3	P
0	1				1/4
1		1			1/4
2			1		1/4
3				1	1/4

Table 3.5: Channels of SUGAR-Length and MILK-Length

Then we merge W_1 and W_2 to construct the *Flavour* - Length channel $W_1 \otimes W_2((y_1, y_2)|(x_1, x_2))$ where $(y_1, y_2) \in l \otimes l, (x_1, x_2) \in D_1 \otimes D_2$:

$W_1 \otimes W_2((y_1, y_2)\|(x_1, x_2))$	(0,0)	(0,1)	...	(y_1, y_2)	...	(3,2)	(3,3)	P
(0,0)	1							1/16
(0,1)		1						1/16
...								
(x_1, x_2)				$P((y_1, y_2)\|(x_1, x_2))$				$P(x_1, x_2)$
...								
(3,2)						1		1/16
(3,3)							1	1/16

Table 3.6: *Flavour*-Length channel

The right end column of probability is simply the joint probability of both inputs of W_1 and W_2.

In this application, degree of SUGAR and degree of MILK are independent variables. This implies their joint probability is simply the product of their marginal probabilities:

$$P(x_1, x_2) = P(x_1)P(x_2) \tag{3.5}$$

Notice that since the output of such channel are actually the length of its input; therefore for a given input, its output is deterministic, i.e.

$$P((y_1, y_2)|(x_1, x_2)) = \begin{cases} 1 & \text{if } y_1 = |x_1| \text{ and } y_2 = |x_2| \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

The merged channel $W_1 \otimes W_2$ results in a table with size of $(|X_1||X_2|)$ rows and $(|Y_1||Y_2|)$ columns. This implies that the merge operation of two channels will potentially has an exponential time and space complexity. However, this can be improved by compressing the channel.

The first thing is that the merged output are actually lengths of both inputs; hence (y_1, y_2) can be replaced by their sum: $y = y_1 + y_2$. Therefore Table 3.6 can be compressed by combing columns with a same length, i.e. we can merge columns into one if $(y_1 + y_2) = (y\prime_1 + y\prime_2)$. The combination is simply the vector sum of two columns

So we can reconstruct $W_1 \otimes W_2$ as:

$(W_1 \otimes W_2)\prime(y = y_1 + y_2 \mid (x_1, x_2))$	0	1	2	3	4	5	6	P
(0,0)	1							1/16
(0,1)		1						1/16
...								
(1,0)		1						
...								
(3,2)						1		1/16
(3,3)							1	1/16

Table 3.7: Compressed *Flavour*-Length channel

In Table 3.7, we can see that different inputs can map to the same length, e.g. $(0, 1)$ and $(1, 0)$ all results to $l = 1$.

Practically, we can further compress this channel with the cost of resolution of input. Generally, there are some facts that worth notice:

- As in (3.6), length is deterministic given a content. Therefore it is a reasonable choice to merge contents which will result into same length.
- The intuition of combing two rows with the same length can be interpreted as follow: for two rows with the same output $W(y|x = x_1)$ and $W(y|x = x_2)$, the merged row represents $W(y|x = x_1 \text{ or } x = x_2)$.
- For such a channel, each input are exclusive events; thus the probability of the input of merged rows is simply the sum of the probability of each row: $P(x_{merged}) = P(x_1) + P(x_2)$

So if we compress Table 3.7 by the same $(y_1 + y_2)$ which is indeed $|Flavour|$, we will have a further compressed $W_1 \otimes W_2$:

$(W_1 \otimes W_2)''(y = y_1 + y_2 \mid x = x_1 + x_2)$	0	1	2	3	4	5	6	P
0	1							1/16
1		1						1/8
2			1					3/16
3				1				1/4
4					1			3/16
5						1		1/8
6							1	1/16

Table 3.8: Further Compressed (with less resolution) *Flavour*-Length channel

The actual degree of SUGAR and MILK are lost in Table 3.8 during this compression, but it also reduced the number of rows from 16[1] to 7.

[1] $(W_1 \otimes W_2)\prime$ has $4 \times 4 = 16$ rows.

By applying the same strategy again to merge the $Order$ channel W_0 with $(W_1 \otimes W_2)$, we will have the $OrderFlavour$-Length channel $W = W_0 \otimes W_1 \otimes W_2$. Then finally as described in Example 3.3.1, we can construct the leakage channel $W^{-1}(x|y)$ (see Appendix A) for the $OrderFlavour$ packets .

To generalise, given the distribution of the plaintext, the leakage channel is constructed as following:

1 If the space of plaintext is large, break the plaintext-length channel into several sub-channels.

2 Compute the sub-channels and compress them. Resolution may lost during the compression.

3 Merge the sub-channels to construct the plaintext-length channel.

4 Use Bayes' Theorem to invert plaintext-length channel.

Another aspect to view such leakage channel is to analyse its capacity, i.e. the maximum mutual information of content and length, as described in [7].

3.3.5 Guessing Plaintext Using Joint Packet Length

In Section 3.3.4 we described a method of packet analysis against a single packet in a session. It is possible to improve the analysis by looking at the packets jointly. As presented in [7], the sequence of packets lengths can be viewed as a vector.

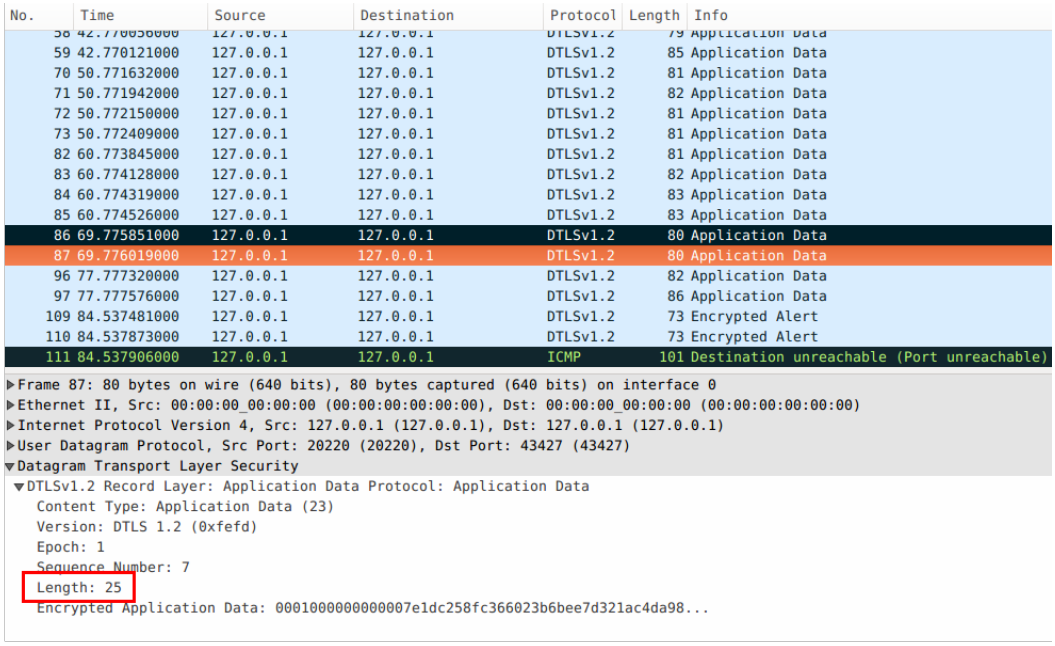

Figure 3.5: Captured Leaky Coffee packets

Example 3.3.3. For example, a 2-packets session (packet No 86 and 87) has been marked out in Figure 3.5. The values of DTLS Length are both 25 as marked in red rectangle. Their actual plaintext length can then be computed as 8 and 8 bytes respectively by Equation (3.1).

It is possible to do the single packet analysis described in Section 3.3.4 on each of these packets. We immediately know that plaintext in the first packet is "ESPRESSO"; whilst the second one could be either "ESPRESSO" or "MOCHA" with a *Flavour* of length 3. However, the analysis of the second packet is in fact unnecessary at all as the application specifies that an "ESPRESSO" *Order* can only be responded with an "ESPRESSO" with NULL *Flavour*.

The application specifies that the first part of the $Order||Flavour$ simply echoes the first packet; therefore in fact we can immediately tell that the second packet is "ESPRESSO".

One way to model this is to "expand" the channel. Instead of using the content at each packet from as the input, we can write them as a vector:

$$\vec{x} =< x_1, x_2, ..., x_n >$$

and similarly, we can also write their length as another vector:

$$\vec{l} =< y_1, y_2, ..., y_n >$$

Then we can use the same strategy in Section 3.3.4 to construct the leakage channel $W^{-1}(X, Y)$ where $\vec{x} \in X$ and $\vec{y} \in Y$.

A problem in this method is the resulting channel is of the size of the Cartesian product of all contents in every packet. However, in this Leaky Coffee application most of the cells is actually 0 which could be used to reduce its storage space; but such optimisation is heavily application dependant.

A probably better way to model this side channel is to use Hidden Markov Model[24] similar to [10]. Techniques of Machine Learning might also be able to utilise this side channel information more efficiently. However, in this "intentionally crafted" Leaky Coffee application, the first packet seems always enough to reveal (roughly) the rest of plaintext in a session.

3.3.6 Estimate plaintext distribution

In Section 3.3.4 and Section 3.3.5 we have demonstrated how to construct a channel given the set of plaintext and their distribution. In this section, we will try to estimate the plaintext distribution by the distribution of ciphertext.

The general idea is that the distribution of length (as presented as P column in Table 3.3) can actually be observed from the ciphertext; therefore we can revert the process and use it to estimate the distribution of plaintext.

Example 3.3.4. Assume we have a sample of encrypted *Order* packet collected where we estimated the its length distribution as following:

y	P
5	d_1
8	d_2
9	d_3

Table 3.9: Estimated length distribution from encrypted *Order* packets

Similar to Example 3.3.1, the first step is to construct a Content-Length channel. The difference is that this time we do not have the pre-knowledge of plaintext distribution; therefore we represent the content distribution as unknown variables p_i for each content.

$W(y\|x)$	5	8	9	P
"AMERICANO"			1	p_1
"CAPPUCINO"			1	p_2
"MOCHA"	1			p_3
"ESPRESSO"		1		p_4

Table 3.10: Content-Length Channel with unknown distibution of *Order*

Their joint distribution follows immediately.

$\widehat{WP}$	P
("AMERICANO",9)	p_1
("CAPPUCINO",9)	p_2
("MOCHA",5)	p_3
("ESPRESSO",8)	p_4

Table 3.11: Joint distribution of $(Order, l)$ with unknown distribution of $Order$

Then we can compute the marginal distribution of length.

y	P
5	p_3
8	p_4
9	$p_1 + p_2$

Table 3.12: Marginal distribution of l with unknown distribution of $Order$

By linking Table 3.12 with Table 3.9 we have the following constrains to the content distribution:

$$\begin{cases} p_3 = d_1 \\ p_4 = d_2 \\ p_1 + p_2 = d_3 \\ \sum_{i=1}^{4} p_i = 1 \end{cases} \tag{3.7}$$

Any solution satisfies Equation (3.7) can be viewed as a reasonable guess to the distribution of the contents.

Chapter 4

Plan

The short term plan is to finish the toy application analysis in Chapter 3. There are these directions in this task:

1. So far we have mainly focused on demonstrating the phenomenon similar to those described in [6] and [7] exists in our set up, but there could possibly be more. For example, it is probably worth study whether it is feasible to deduce the plaintext distribution in Section 3.2 given encrypted packets.

2. To see if the attacks in Section 3.2 can be optimised.

3. Apply and evaluate some countermeasures to these attacks.

4. Only a few, nearly none, attempt has been made to study types of attack other than [6] and [7]; hence this might also be worth trying.

One of the greatest deficiency in the work so far is the risk of being alienated from reality since the toy applications are designed in an extremely abstracted way whilst the real world application could be much more complicated and will not be intentionally designed to be vulnerable to attacks. Therefore as a long term plan, it is very important to get in touch with some actual application and evaluate those we have developed from the toys. Having said so, one of the greatest challenges is the immaturity of IoT; hence following the trend of related technologies would be an important and constant task in this project.

Chapter 5

Other Activities

I have been one of the TAs for Cryptography A and System Security during the first term of 2014 - 2015.

I also attended Real World Crypto 2015.

Appendix A

OrderFlavour-Length leakage channel

In this application, the joint probability of *Order* and *Flavour* are simply the product of their marginal probability. However, since "ESPRESSO" will always followed by *Flavour* of of both degree of SUGAR and MILK being 0(see Section 3.2.1); hence

$$P(x_1, x_2|\text{“ESPRESSO”}) = \begin{cases} 1 & \text{if } x_1 = x_2 = 0 \\ 0 & \text{otherwise} \end{cases}$$

Therefore

$$P(\text{“ESPRESSO”}, x_1, x_2) = \begin{cases} 1/4 & \text{if } x_1 = x_2 = 0 \\ 0 & \text{otherwise} \end{cases}$$

Bibliography

[1] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007. ISBN: 978-0-387-30857-9.

[2] Daniel J. Bernstein. "Cache-timing attacks on AES". In: (2004). URL: http://cr.yp.to/papers.html#cachetiming.

[3] Dag Arne Osvik, Adi Shamir, and Eran Tromer. "Cache attacks and Countermeasures: the Case of AES." In: *IACR Cryptology ePrint Archive* 2005 (2005), p. 271. URL: http://dblp.uni-trier.de/db/journals/iacr/iacr2005.html#OsvikST05.

[4] International Organization for Standardization ISO. *ISO/IEC 7498-1Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model*. Tech. rep. June 1994, pp. 34–65+.

[5] *tinyDTLS*. http://tinydtls.sourceforge.net/.

[6] Shuo Chen et al. "Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow". In: *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*. 2010, pp. 191–206. DOI: 10.1109/SP.2010.20. URL: http://doi.ieeecomputersociety.org/10.1109/SP.2010.20.

[7] Luke Mather and Elisabeth Oswald. "Pinpointing side-channel information leaks in web applications". In: *J. Cryptographic Engineering* 2.3 (2012), pp. 161–177. DOI: 10.1007/s13389-012-0036-0. URL: http://dx.doi.org/10.1007/s13389-012-0036-0.

[8] Mário S. Alvim et al. "Measuring Information Leakage Using Generalized Gain Functions". In: *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*. 2012, pp. 265–279. DOI: 10.1109/CSF.2012.26. URL: http://doi.ieeecomputersociety.org/10.1109/CSF.2012.26.

[9] Tom Chothia and Apratim Guha. "A Statistical Test for Information Leaks Using Continuous Mutual Information". In: *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF 2011, Cernay-la-Ville, France, 27-29 June, 2011*. 2011, pp. 177–190. DOI: 10.1109/CSF.

2011.19. URL: http://doi.ieeecomputersociety.org/10.1109/CSF.2011.19.

[10] George Danezis. *Traffic Analysis of the HTTP Protocol over TLS*.

[11] Alexander Schaub et al. "Attacking Suggest Boxes in Web Applications Over HTTPS Using Side-Channel Stochastic Algorithms". In: *IACR Cryptology ePrint Archive* 2014 (2014), p. 959. URL: http://eprint.iacr.org/2014/959.

[12] John Kelsey. "Compression and Information Leakage of Plaintext". In: *Fast Software Encryption, 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002, Revised Papers*. 2002, pp. 263–276. DOI: 10.1007/3-540-45661-9_21. URL: http://dx.doi.org/10.1007/3-540-45661-9_21.

[13] *The CRIME Attack*. https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU/edit#slide\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{i\global\mathchardef\accent@spacefactor\spacefactor}\accent22i\egroup\spacefactor\accent@spacefactord.g1d134dff_1_222.

[14] Yoel Gluck, Neal Harris, and Angelo (Angel) Prado. *Breach: Reviving the CRIME Attack*. Tech. rep. 2013. URL: http://breachattack.com/resources/BREACH%20-%20SSL,%20gone%20in%2030%20seconds.pdf.

[15] Janaka Alawatugoda, Douglas Stebila, and Colin Boyd. "Protecting Encrypted Cookies from Compression Side-Channel Attacks". In: *IACR Cryptology ePrint Archive* 2014 (2014), p. 724. URL: http://eprint.iacr.org/2014/724.

[16] Serge Vaudenay. "Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS ..." In: *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*. 2002, pp. 534–546. DOI: 10.1007/3-540-46035-7_35. URL: http://dx.doi.org/10.1007/3-540-46035-7_35.

[17] Nadhem J. AlFardan and Kenneth G. Paterson. "Lucky Thirteen: Breaking the TLS and DTLS Record Protocols". In: *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*. 2013, pp. 526–540. DOI: 10.1109/SP.2013.42. URL: http://dx.doi.org/10.1109/SP.2013.42.

[18] J. Postel. *Internet Protocol*. RFC 791 (INTERNET STANDARD). Updated by RFCs 1349, 2474, 6864. Internet Engineering Task Force, Sept. 1981. URL: http://www.ietf.org/rfc/rfc791.txt.

[19] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460 (Draft Standard). Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112. Internet Engineering Task Force, Dec. 1998. URL: http://www.ietf.org/rfc/rfc2460.txt.

[20] J. Hui and P. Thubert. *Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*. RFC 6282 (Proposed Standard). Internet Engineering Task Force, Sept. 2011. URL: http://www.ietf.org/rfc/rfc6282.txt.

[21] E. Rescorla and N. Modadugu. *Datagram Transport Layer Security Version 1.2*. RFC 6347 (Proposed Standard). Internet Engineering Task Force, Jan. 2012. URL: http://www.ietf.org/rfc/rfc6347.txt.

[22] Z. Shelby, K. Hartke, and C. Bormann. *The Constrained Application Protocol (CoAP)*. RFC 7252 (Proposed Standard). Internet Engineering Task Force, June 2014. URL: http://www.ietf.org/rfc/rfc7252.txt.

[23] D. McGrew et al. *AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS*. RFC 7251 (Informational). Internet Engineering Task Force, June 2014. URL: http://www.ietf.org/rfc/rfc7251.txt.

[24] Leonard E. Baum and Ted Petrie. "Statistical Inference for Probabilistic Functions of Finite State Markov Chains". In: *Ann. Math. Statist.* 37.6 (Dec. 1966), pp. 1554–1563. DOI: 10.1214/aoms/1177699147. URL: http://dx.doi.org/10.1214/aoms/1177699147.