Contents

Chapter 1

Theoretical background

In this chapter we will provide the necessary background for the user to understand the mechanisms used later in the paper. The description of the following systems is a brief introduction, intended to familiarize the reader with concepts that are fundamental for the methods presented.

Specifically, section 1.1 describes the functionality of the gzip compression method and the algorithms that it entails. Section **??** covers the same-origin policy that applies in the web application security model. In section **??** we explain the Transport Layer Security, which is the main protocol used today to provide communications security over the Internet. Finally, in section **??** we describe attack methodologies in order for an adversary to perform a Man-in-the-middle attack, such as ARP spoofing and DNS poisoning.

1.1 gzip

gzip is a software application used for file compression and decompression. It is the most used encryption method on the Internet, integrated in protocols such as the Hypertext Transfer Protocol (HTTP), the Extensible Messaging and Presence Protocol (XMPP) and many more. Derivatives of gzip include the tar utility, which can extract .tar.gz files, as well as zlib, an abstraction of the DEFLATE algorithm in library form.[1]

It is based on the DEFLATE algorithm, which is a combination of LZ77 and Huffman coding. DEFLATE could be described by the following encryption schema:

$DEFLATE(m) = Huffman(LZ77(m))$

In the following sections we will briefly describe the functionality of both these compression algorithms.

1.1.1 LZ77

LZ77 is a lossless data compression algorithm published in paper by A. Lempel and J. Ziv in 1977. [1] It achieves compression by replacing repeated occurences of data with references to a copy of that data existing earlier in the uncompressed data stream. The reference composes of a pair of numbers, the first of which represents the length of the

[1] https://en.wikipedia.org/wiki/Gzip

repeated portion, while the second describes the distance backwards in the stream, until the beginning of the portion is met. In order to spot repeats, the protocol needs to keep track of some amount of the most recent data, specifically the 32 Kb latest. This data is held in a sliding window, so, in order for a portion of data to be compressed, the initial appearance of this repeated portion needs to have occurred at most 32 Kb up the data stream. Also, the minimum legth of a text to be compressed is 3 characters, while compressed text can also have literals as well as pointers.

Below you can see an example of a step-by-step execution of the algorithm for a specific text:

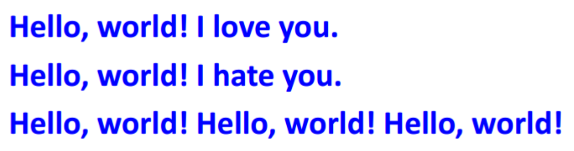
Hello, world! I love you.
Hello, world! I hate you.
Hello, world! Hello, world! Hello, world!

Figure 1.1: First we get the plaintext to be compressed.

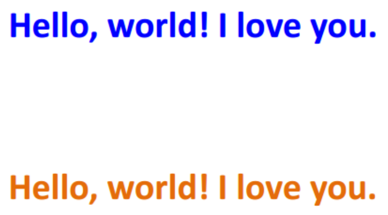

Figure 1.2: Compression starts with literal representation.

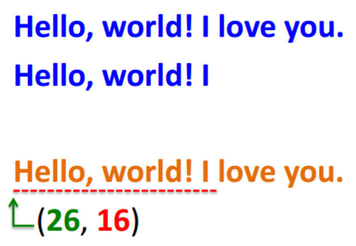

Figure 1.3: We then use a pointer at distance 26 and length 16.

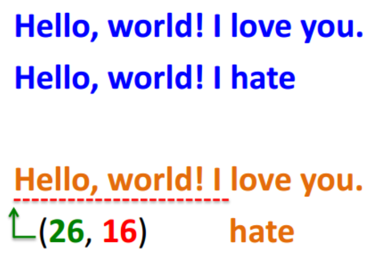

Figure 1.4: We continue with literal.

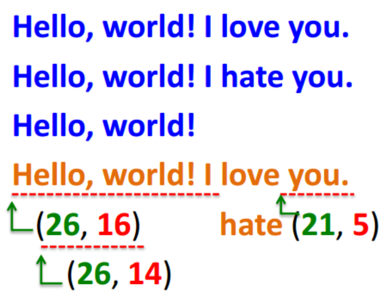
Hello, world! I love you.
Hello, world! I hate you.
Hello, world!
Hello, world! I love you.
(26, 16)
hate (21, 5)
(26, 14)

Figure 1.5: We use a pointer pointing to a pointer.

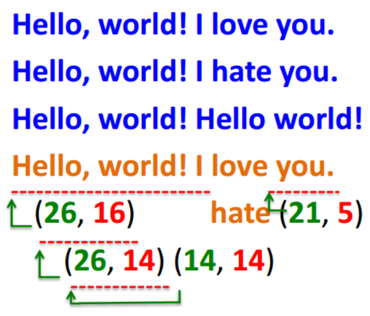
Hello, world! I love you.
Hello, world! I hate you.
Hello, world! Hello world!
Hello, world! I love you.
(26, 16)
hate (21, 5)
(26, 14) (14, 14)

Figure 1.6: We then use a pointer pointing to a pointer pointing to a pointer.

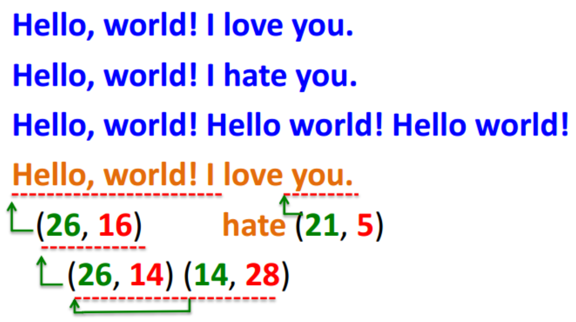
Hello, world! I love you.
Hello, world! I hate you.
Hello, world! Hello world! Hello world!
Hello, world! I love you.
(26, 16)
hate (21, 5)
(26, 14) (14, 28)

Figure 1.7: Finally, we use a pointer pointing to itself.

Chapter 2

Partially Chosen Plaintext Attack

Traditionally, cryptographers have used games for security analysis. Such games include the indistinguishability under chosen-plaintext-attack (IND-CPA), the indistinguishability under chosen ciphertext attack/adaptive chosen ciphertext attack (IND-CCA1, IND-CCA2) etc[1]. In this chapter, we introduce a definition for a new property of encryption schemes, called indistinguishability under partially-chosen-plaintext-attack (IND-PCPA). We will also show provide comparison between IND-PCPA and other kwown forms of cryptosystem properties.

2.1 Partially Chosen Plaintext Indistinguishability

2.1.1 Definition

IND-PCPA uses a definition similar to that of IND-CPA. For a probabilistic asymmetric key encryption algorithm, indistinguishability under partially chosen plaintext attack (IND-PCPA) is defined by the following game between an adversary and a challenger.

- The challenger generates a pair P_k, S_k and publishes P_k to the adversary.
- The adversary may perform a polynomially bounded number of encryptions or other operations.
- Eventually, the adversary submits two distinct chosen plaintexts M_0, M_1 to the challenger.
- The challenger selects a bit $b \in 0, 1$ uniformly at random.
- The adversary can then submit any number of selected plaintexts $R_i, i \in N, |R| \geq 0$, so the challenger sends the ciphertext $C_i = E(P_k, M_b || R_i)$ back to the adversary.
- The adversary is free to perform any number of additional computations or encryptions, before finally guessing the value of b.

A cryptosystem is indistinguishable under partially chosen plaintext attack, if every probabilistic polynomial time adversary has only a negligible advantage on finding b over random guessing. An adversary is said to have a negligible advantage if it wins

[1] https://en.wikipedia.org/wiki/Ciphertext_indistinguishability

the above game with probability $\frac{1}{2} + \epsilon(k)$, where $\epsilon(k)$ is a negligible function in the security parameter k.

Intuitively, we can think that the adversary has the ability to modify the plaintext of a message, by appending a portion of data of his own choice to it, without knowledge of the plaintext itself. He can then acquire the ciphertext of the modified text and perform any kinds of computations on it. A system could then be described as IND-PCPA, if the adversary is unable to gain more information about the plaintext, than he could by guessing at random.

2.1.2 IND-PCPA vs IND-CPA

Suppose the adversary submits the empty string as the chosen plaintext, a choice which is allowed by the definition of the game. The challenger would then send back the ciphertext $C_i = E(P_k, M_b||\text{" "}) = E(P_k, M_b)$, which is the ciphertext described in the IND-CPA game. Therefore, if the adversary has the ability to beat the game of IND-PCPA, i.e. if the system is not indistinguishable under partially chosen plaintext attacks, he also has the ability to beat the game of IND-CPA. Thus we have shown that IND-PCPA is at least as strong as IND-CPA. The above intuitive proof could be expressed formally in future works.

2.2 PCPA on compressed encrypted protocols

2.2.1 Compression-before-encryption and vice versa

When having a system that applies both compression and encryption on a given plaintext, it would be interesting to investigate the order the transformations should be executed.

Lossless data compression algorithms rely on statistical patters to reduce the size of the data to be compressed, without losing information. Such a method is possible, since most real-world data has statistical redundancy. However, it can be understood from the above that such compression algorithms will fail to compress some data sets, if there is no statistical pattern to exploit.

Encryption algorithms rely on adding entropy on the ciphertext produced. If the ciphertext contains repeated portions or statistical patterns, such behaviour can be exploited to deduce the plaintext.

In the case that we apply compression after encryption, the text to be compressed should demostrate no statistical analysis exploits, as described above. That way compression will be unable to reduce the size of the data. In addition, compression after encryption does not increase the security of the protocol.

On the other hand, applying encryption after compression seems a better solution. The compression algorithm can exploit the statistical redundancies of the plaintext, while the encryption algorithm, if applied perfectly on the compressed text, should produce a random stream of data. Also, since compression also adds entropy, this scheme should make it harder for attackers who rely on differential cryptanalysis to break the system.

2.2.2 PCPA scenario on compression-before-encryption protocol

Let's assume a system that composes encryption and compression in the following manner:

$c = Encrypt(Compress(m))$

where c is the ciphertext and m is the plaintext.

Suppose the plaintext contains a specific secret, among random strings of data, and the attacker can issue a PCPA with a chosen plaintext, which we will call reflection. The plaintext then takes the form:

$m = n_1 || secret || n_2 || reflection || n_3$

where n_1, n_2, n_3 are random nonces.

If the reflection is equal to the secret, the compression mechanism will recognize the pattern and compress the two portions. In other case, the two strings will not demonstrate any statistical redundancy and compression will perform worse. As a result, in the first case the data to be encrypted will be smaller than in the second case, thus demonstrating a pattern that the attacker can exploit.

Bibliography

[1] Abraham Ziv, Jacob; Lempel. A universal algorithm for sequential data compression. Information Theory, IEEE Transactions, 23:337–343.