SafeDeflate: compression without leaking secrets

Michał Zieliński

Abstract

CRIME[1] and BREACH[2] attacks on TLS/SSL leverage the fact that compression ratio is not hidden by encryption to recover content of secrets. We introduce SafeDeflate—a modification of a standard Deflate algorithm which compression ratio does not leak information about secret tokens. The modification is compatible with existing Deflate and gzip decompressors. We introduce a model in which attacker can obtain ciphertexts of arbitrary compressed plaintext containing secret values. Then we prove that SafeDeflate is secure in this model.

1 Introduction

Various strategies[3] has been proposed for preventing the BREACH and CRIME attacks [4]. Adding random amount of padding after compression is one of them. This is not good enough—with uniform 16 byte padding determining unpadded length with 99% certainty requires only 200 queries (and the number of tries scales roughly quadratically). The strategy used by the CRIME attack can be viewed as a binary search, so the amount of required queries can probably be decreased by treating the problem as a binary search with lies (Ulam problem [5]).

The only effective solutions are either disabling compression entirely or taking care to never repeat secret tokens at application level [3]. They are obviously unsatisfactory.

In this paper we introduce a model called *secrets in chosen container attack*. In this model the attacker can ask the oracle to return encrypted ciphertext of the compressed message composed by him. The message consists of secret tokens unknown to attacker interleaved by arbitrary text chosen by the attacker. We assume that underlying cipher is secure and after each query, the attacker only learns length of the compressed message.

Afterwards we describe SafeDeflate—a modification of standard Deflate compressor. If we assume that secrets are drawn from small alphabet (e.g. alphanumeric characters), a simple modification suffices to maintain both good compression ratio and security. We also show more general algorithm, which is not restricted to small alphabets.

2 Attack model

We model *secrets in chosen container attack* as a tuple $(\Sigma_A, \Sigma_S, \texttt{SECRETLEN}, \texttt{MSGLEN}, \texttt{MIN}\Sigma)$, where:

- Σ_A is a text alphabet (normally $0 \ldots 255$)
- $\Sigma_S \subset \Sigma_A$ is a secret alphabet
- for each alphabet $\Sigma \subset \Sigma_S$, each length k ($k \geq \texttt{SECRETLEN}$) and any integer j, secret $S_{\Sigma,k,j}$ is a random variable. This random variable has uniform distribution over all strings of length k over Σ. For any Σ, k and j, $S_{\Sigma,k,j}$ are mutually independent. Alphabets must have size of at least $\texttt{MIN}\Sigma$.

Let `ORACLE` be the oracle function, returning length of compressed message x in stream[1] i. Each stream maintains its own state of compression algorithm. We assume that attacker is a probabilistic Turing machine equipped with oracle `ORACLE`.[2] Before the attacker starts running, the secrets are drawn (an elementary event ω is chosen from the sample space).

Attacker may ask oracle to return the length of the compressed text x prepared by the attacker separated by secret tokens. The parameter x is in form $\mathrm{val}(w_1)\,\mathrm{val}(w_2)\ldots\mathrm{val}(w_n)$ where w_i is either a character from Σ_A or a tuple (Σ_1, k_1, j_1) (where S_{Σ_1,k_1,j_1} is a secret). Let

$$\texttt{ORACLE}(i, x) = \text{length of } \mathrm{val}(w_1)\,\mathrm{val}(w_2)\ldots\mathrm{val}(w_n) \text{ after compression}$$

where $\mathrm{val}(w_i) = w_i$ if $w_i \in \Sigma_A$ or $\mathrm{val}((\Sigma_1, k_1, j_1)) = S_{(\Sigma_1,k_1,j_1)}$ otherwise. In addition, all queries must satisfy $|\,\mathrm{val}(w_1)\,\mathrm{val}(w_2)\ldots\mathrm{val}(w_n)| < \texttt{MSGLEN}$.

Note that the value $\texttt{ORACLE}(i, x)$ is a random variable (as it depends on secrets)—the attacker gets the actual value according to which elementary event ω was chosen at the start.

Each random variable $S_{\Sigma,k,j}$ represents a secret string, which attacker tries to guess. Initially, the attacker knows nothing about them (apart from the fact they are independent and uniform). When he receives responses from the oracle, he learns more about the secrets—the conditional probability distribution of $S_{\Sigma,k,j}$, given received oracle responses, changes.

We say that the compression algorithm is (p, l)-*secure* if attacker cannot guess any secret $S_{\Sigma,k,j}$ with probability greater than p making l queries to the oracle.

3 The SafeDeflate algorithm

3.1 LZ77

LZ77 [6] algorithm keeps track of last `BUFLEN` (typically 32 kB) of compressed data in a *compression buffer* (usually called dictionary). When new byte comes, the compressor checks if current suffix occurs somewhere else in a dictionary. If so, the algorithm may output reference to previous occurrence instead of inserting

[1] Most compression algorithms process data in streams i.e. contests of compressed packet depends on previous data.

[2] We actually do not need the attacker to be computationally bounded.

the bytes verbatim (we say that this suffix is *matched*). Standard Deflate implementations choose to do that when the suffix exceeds some predefined length (other criteria are also employed).

The problem with this approach is the fact that small parts of secrets can be matched, resulting in different compressed length depending on attacker guesses:

In the following example, boxed part is a matched suffix which will be outputted as a reference, the rest of text will be outputted verbatim.

```
<input value="yo5aaaaaa"...<input value="yo5Gee6keiqu9oona"...
```

We can see that if attacker can control the first part of the plaintext, he may try to guess the secret. The length of ciphertext (or length of compressed plaintext) will depend on how many characters from secret he has already guessed. Using this information, the attacker can guess whole secret in linear time.

Informally, if the boundary of the matched suffix can be crossed by a secret in an arbitrary place, the compression algorithm will be vulnerable to CRIME-like attacks. Note that it is not a problem for a matched suffix to contain a secret as a whole.

SafeDeflate LZ77 matches longest suffix S that:

- is longer than some predefined constant (e.g. it is not useful to match 1 byte suffixes) and
- has form $S = awb$, where either
  - $a, b \notin \Sigma_S$ or
  - $a, b \in \texttt{DICT}, |x| > C_1$, where `DICT` is a predefined set of words that are *long enough* and C_1 is a *big enough* constant (this ensures that *most* secrets can't be crossed by a matched suffix)

Compression buffer for each stream is defined as the concatenation of previous messages.

3.2 Huffman encoding

The LZ77 phase outputs a sequence of symbols. The symbols can be of two types [6]:

- $0 \ldots 255$ for verbatim text,
- additional symbols for encoding references.

The string is then processed using Huffman encoding—that is, a prefix code is constructed for it and the string is encoded using it. The compression algorithm may choose to either use predefined static table, compute optimal table or use table from previous block [6].

4 Analysis

The length of string after Huffman encoding is a function of frequencies of symbols in LZ77 encoded string (and frequencies of symbols in previous blocks). It follows that the length is also a function of frequencies of strings outputted verbatim and offsets of references.

Thus, if our algorithm is secure against an attacker that learns strings outputted verbatim and offsets of references, it is also secure against attacker that only learns the length.

4.1 Properties of the dictionary

Let $\texttt{DICT}'$ be a closure of DICT over "overlapping concatenation"—that is, $\texttt{DICT}'$ is a minimal set containing DICT and for which $w \in \texttt{DICT}'$ if there exists $x, y \in \texttt{DICT}'$ s.t. x is a prefix of w, y is a suffix of w and $|x| + |y| \geq |w|$.

For a word w, we say that position $i \in \{0, \ldots, |w| - 1\}$ is *marked* iff there is $v \in \texttt{DICT}$, s.t. v is a subword of w and it overlaps with position i in word w. Every maximal marked subword is in $\texttt{DICT}'$—this follows from the construction of $\texttt{DICT}'$.

For a word w, we say that position $i \in \{0, \ldots, |w|-1\}$ is *ps-marked* (*prefix-suffix marked*) iff there exist words $x, y \in \Sigma^*$ s.t. position $i + |x|$ is marked in xwy.

We assume that all words in DICT have same length m.

Lemma 1a If a word $w \in \texttt{DICT}'$, it is an overlapping concatenation of at most $2|w|/m$ words from DICT.

Proof By induction on $|w|$. For $|w| = m$, obviously $w \in \texttt{DICT}$. For $|w| > m$, let i_1 be the position of last occurrence of word from DICT in w, s.t $i_1 \leq m$. Let i_2 be position of first occurrence of word from DICT in w, s.t $i_2 > m$.

- If i_2 exists then $w_{i_2 \ldots}$ is overlapping concatenation of at most $2(|w| - m)/m = 2|w|/m - 2$ words[3]. As w is overlapping concatenation of $w_{0 \ldots m}$, $w_{i_1 \ldots i_1 + m}$ and $w_{i_2 \ldots}$, the Lemma holds.
- Otherwise $|w| = i_1 + m$ and the Lemma holds.

Lemma 1b There are at most $2(|\texttt{DICT}|m)^{2s/m}$ words of length s in $\texttt{DICT}'$.

Proof Let S_k be the set of words which are overlapping concatenations of k words. First, we will prove by induction, that $|S_k| \leq (|\texttt{DICT}|m)^k$.

Obviously $|S_1| = |\texttt{DICT}| < 2|\texttt{DICT}|m$. For $k > 1$, any word is overlapping concatenation of one of words from $S_1 = \texttt{DICT}$ and from S_{k-1}. As the former word has length equal to m, this concatenation can be done in m ways. Then $|S_k| \leq m \cdot |\texttt{DICT}| \cdot |S_{k-1}| = (|\texttt{DICT}|m)^k$.

Word of length s is an overlapping concatenation of at most $2|w|/m$ words, so there are at most $\sum_{i=1}^{2s/m} (|\texttt{DICT}|m)^i < 2(|\texttt{DICT}|m)^{2s/m}$ words of length s in $\texttt{DICT}'$.

[3] By $x_{a \ldots b}$ we denote a subword of x starting at a with length $b - a$. Let $x_{a \ldots} = x_{a \ldots |w|}$.

Lemma 1c There are at most $2(|\texttt{DICT}|m)^{2s/m+2}$ words of length s that are prefix of a word in $\texttt{DICT}'$.

Proof If word w is a prefix of $w' \in \texttt{DICT}'$, then there exists $w'' \in \texttt{DICT}'$ s.t. $|w''| \leq |w| + m$ and w is prefix of w''. Then, by Lemma 1a w'' is overlapping concatenation of at most $2|w|/m + 2$. Then the bound for this Lemma is then constructed analogously to bound from Lemma 1b.

Lemma 1: For every $L, s \in \mathbb{N}$, $s \leq L$, if X is a random word uniformly sampled from Σ^L, the following holds:

1. In word X first s positions are ps-marked with probability $\leq (|\texttt{DICT}|m)^{2s/m+2}/|\Sigma|^s$.
2. All marked characters in X form a subword of length $\geq s$ with probability $\leq L(|\texttt{DICT}|m)^{2s/m+2}/|\Sigma|^s$.
3. There are two marked characters in X that are not part of a marked subword with probability $\leq (L|\texttt{DICT}|/|\Sigma|^m)^2$.

Proof:

1. If first s positions are ps-marked, then the word $X_{0\ldots s}$ is a prefix of some word $w \in \texttt{DICT}'$. By Lemma 1c, there are at most $|\Sigma|^{L-s} \cdot (|\texttt{DICT}|m)^{2s/m+2}$ such words.
2. If marked characters in X form a subword of length $\geq s$, then there exists position i s.t. $X_{i\ldots i+s}$ is a prefix of word in $\texttt{DICT}'$. The position i can be chosen in L ways. Then by Lemma 1c, there are at most $|\Sigma|^{L-s} \cdot L \cdot (|\texttt{DICT}|m)^{2s/m+2}$ such words.
3. There are at least two disjoint subwords of X that are in $\texttt{DICT}$. Each of these words can be chosen in at most L ways, so there are at most $(L|\texttt{DICT}|)^2|\Sigma|^{L-2m}$ such words.

4.2 Token character frequencies

Let $\texttt{samefreq}(w)$ be a set of permutations of word w. That is, let $\texttt{samefreq}(w) = \{w' \in \Sigma^n | \forall c \in \Sigma, w' \in \texttt{samefreq}(w) : |w|_c = |w'|_c\}$.

Lemma 2: Let Σ be an alphabet and let $|\Sigma|$ be a power of two. For every $n \in \mathbb{N}$, if w is a random word uniformly sampled from Σ^n then $|\texttt{samefreq}(w)| \geq (1.14^{\log|\Sigma|})^n$ with probability at least $1 - 2 \cdot (1.14^{\log|\Sigma|})^n$.

Proof:

Without loss of generality let $\Sigma = \{0,1\}^k$. For any word $w \in \Sigma^n$, by $w_{i,j}$ we denote j-th bit of the symbol w_i. Let $W_j \in \{0,1\}^n$ be defined as $W_{j,i} = w_{i,j}$ (i.e. sequence of j-th bits of symbols from w). Obviously $|\texttt{samefreq}(w)| = \prod_j |\texttt{samefreq}(W_j)|$.

The number of occurrences of character $c \in \{0,1\}$ in word W_j, $|W_j|_c$ has binomial distribution with parameter $\frac{1}{2}$.

By Hoeffding inequality we have (for any $\alpha > 2$ and $c \in \{0,1\}$):

$$P\left(|W_j|_c \leq \tfrac{1}{\alpha}n\right) \leq \exp\left(-2 \cdot n\left(\tfrac{1}{2} - \tfrac{1}{\alpha}\right)^2\right)$$

With probability at least $1 - 2\exp\left(-2 \cdot n\left(\frac{1}{2} - \frac{1}{\alpha}\right)^2\right)$ we have

$$\frac{1}{\alpha} \leq |W_j|_0 \leq 1 - \frac{1}{\alpha}$$

In this case

$$\texttt{samefreq}(W_j) \geq \binom{n}{\frac{n}{\alpha}} \geq \left(\frac{n}{\frac{n}{\alpha}}\right)^{\frac{n}{\alpha}} = \left(\alpha^{1/\alpha}\right)^n$$

Setting $\alpha = 7.5$ we get

$$P(|\texttt{samefreq}(W_j)| \leq 1.3^n) \leq 2 \cdot 1.3^{-n}$$

and

$$P(\forall j \in \{1, \ldots, k/2\} : |\texttt{samefreq}(W_j)| \leq 1.3^n) \leq 2 \cdot 1.3^{-n \cdot k/2}$$

If the latter event does not occur, then for at least $k/2$ strings $|\texttt{samefreq}(W_i)| \geq 1.3^n$, so $\texttt{samefreq}(w) \geq 1.3^{nk/2} \approx 1.14^{nk}$.

Note that this bound is far from being tight.

4.3 Good tokens

For any word $x \in \Sigma^L$ define:

- $L(x)$, $W(x)$ and $R(x)$ as words s.t. $x = L(x)W(x)R(x)$. $W(x)$ is the maximal marked substring s.t. $L(x) \neq \epsilon$ and $R(x) \neq \epsilon$
- $L_1(x)$ as the first $2C_2$ characters of $L(x)$ and $L(x) = L_1(x)L_2(x)$.
- $R_1(x)$ as the last $2C_2$ characters of $R(x)$ and $R(x) = R_2(x)R_1(x)$.

We call $L(x)$, $W(x)$ and $R(x)$ respectively left, center and right side of the word x.

We call a word $x \in \Sigma^L$ *good* if it satisfies the following criteria:

- $L \geq \texttt{SECRETLEN}$
- no prefix nor suffix longer than $2C_2$ characters is ps-marked.
- apart from prefixes and suffixes there is at most one marked nonempty substring w and $|w| \leq 2C_2$
- $|\texttt{samefreq}(L_1(x))| \geq C_4$ or $|\texttt{samefreq}(R_1(x))| \geq C_4$

where $C_4 = (1.14^{\log|\texttt{MINΣ}|})^{(\texttt{SECRETLEN}-4C_2-1)/2}$.

Lemma 3: If X is a random word uniformly sampled from Σ^L, it is good with probability $\geq 1 - 2(|\texttt{DICT}|m)^{4C_2/m+2}/|\Sigma|^{2C_2} - L(|\texttt{DICT}|m)^{4C_2/m+2}/|\Sigma|^{2C_2} - (L|\texttt{DICT}|/|\Sigma|^m)^2 - 2 \cdot (1.14^{\log|\texttt{MINΣ}|})^{-(L-4C_2-1)/2} \cdot L$.

Proof

By Lemma 1 (case 1), the first condition is not satisfied with probability $P_1 \leq 2(|\texttt{DICT}|m)^{4C_2/m+2}/|\Sigma|^{2C_2}$.

By Lemma 1 (cases 2 and 3), the second condition is not satisfied with probability $P_2 \leq L(|\texttt{DICT}|m)^{4C_2/m+2}/|\Sigma|^{2C_2} - (L|\texttt{DICT}|/|\Sigma|^m)^2$.

Consider all partitions of X into $L_1(X)L_2(X)W(X)R_1(X)R_2(X)$. There are L of them and in every one, either $R_2(X)$ or $L_2(X)$ is greater than $(L -$

$4C_2 - 1)/2$. Let P_3 be probability that all these partitions satisfy condition $|\texttt{samefreq}(L(X))| \geq C_4$ or $|\texttt{samefreq}(R(X))| \geq C_4$. By Lemma 2, we know that $P_3 \leq 2 \cdot (1.14^{\log|\texttt{MINΣ}|})^{-(L-4C_2-1)/2} \cdot L$.

Therefore probability of all of these conditions being satisfied is greater than $1 - (P_1 + P_2 + P_3)$.

4.4 The fooling sets

We are going to analyze security of our algorithm by maintaining a set of possible secrets that would produce same outputs of the oracle given specific queries. We call this set *a fooling set* $F^Q_{\Sigma,k,j}$ (as it is often done in communication complexity). Given a sequence of past queries Q we maintain one fooling set $F^Q_{\Sigma,k,j}$ for each secret token $S_{\Sigma,k,j}$. Note that $F^Q_{\Sigma,k,j}$ is a random variable and $S_{\Sigma,k,j} \in F^Q_{\Sigma,k,j}$.

Without loss of generality we may assume that $|S_{\Sigma,k,j}| = \texttt{SECRETLEN}$—any attacker strategy using tokens of greater lengths could be turned into strategy using only SECRETLEN (instead of asking for $S_{\Sigma,k',j}$, ask for $S_{\Sigma,k,j}$ padded with $k' - k$ zeros).

Claim 1: If current fooling set has size $p = |F^Q_{\Sigma,k,j}|$, it is not possible to guess the secret with probability greater than $1/p$. In other words, for a sequence of oracle queries Q and results O, there is no random variable X, s.t. X is independent from $S_{\Sigma,k,j}$ assuming $\texttt{ORACLE}(Q) = O$ and $P(X = S_{\Sigma,k,j}|\texttt{ORACLE}(Q) = O) > \frac{1}{|F_{\Sigma,k,j^Q}|}$.

Initially we set[4]

$$F^{()}_{\Sigma,k,j} := \texttt{samefreq}(L(S_{\Sigma,k,j}))\{W(S_{\Sigma,k,j})R(S_{\Sigma,k,j})\}$$

wherever $|L(S_{\Sigma,k,j})| > |R(S_{\Sigma,k,j})|$ or otherwise

$$F^{()}_{\Sigma,k,j} := \{L(S_{\Sigma,k,j})W(S_{\Sigma,k,j})\}\texttt{samefreq}(R(S_{\Sigma,k,j}))$$

This is obviously a fooling set (there were no queries yet).

Lemma 4: For good tokens, with probability $1 - \frac{1}{|F^Q_{\Sigma,k,j}|}$ after each query to the oracle, the size of the fooling set decreases by not more than $\texttt{BUFLEN} + \texttt{MSGLEN}$.

Proof:

For sake of brevity we will assume that $|L(S_{\Sigma,k,j})| > |R(S_{\Sigma,k,j})|$ (argument for the other case is identical).

After issuing queries Q, the attacker issues an additional query $w = w_1 w_2 \ldots w_n$ to the oracle. Q' is Q with this query appended.

Let A be a set of all substrings of length $|L_2(S_{\Sigma,k,j})|$ of compression buffer after queries Q'. We will show that

$$F^{Q'}_{\Sigma,k,j} = F^Q_{\Sigma,k,j} \setminus (\{L_1(S_{\Sigma,k,j})\}A\{W(S_{\Sigma,k,j})R(S_{\Sigma,k,j})\}) \cup \{S_{\Sigma,k,j}\}$$

[4]If A and B are sets of words, then $AB = \{ab : a \in A, b \in B\}$

is a fooling set with probability $\geq 1 - \frac{1}{|F^Q_{\Sigma,k,j}|}$.

As we have noted before, we can assume that after each query to oracle, the attacker learns:

- character frequencies of strings outputted verbatim (Fr)
- positions and lengths of references (ref_i)

For a fixed ref_i, for every item in the fooling set, Fr is the same (every word in the fooling set has the same character frequencies).

Consider information about token $S_{\Sigma,k,j}$ attacker gets from ref_i:

- for each i-th character outputted verbatim, attackers learns that there is no substring of a compression buffer that is equal to the substring $\text{val}(w_{m\ldots i})$ (for $i > m$) and satisfies SafeDeflate requirements.
- for each ref_i, attacker learns that a substring of a compression buffer is equal to a location of our query and satisfies SafeDeflate requirements.

We can easily see that the first condition is satisfied by strings from the fooling set. If we replace every $S_{\Sigma,k,j}$ (in a query and in a compression buffer) with a string from a fooling set, every substrings that was not equal is still not equal (simply because none of words in a fooling set is a subword of the new compression buffer).

Claim 2: For any nonempty word v and $n \in \mathbb{N}$, there exists exactly one word x s.t. $|x| = n$ and x is a prefix of vX. $\square$

For reference ref_i attacker learns that some subwords of new compression buffer v and v' are equal. Let N and N' be sets of integers. The secret $S_{\Sigma,k,j}$ occurs respectively at positions N and N' in these strings (we let position be negative, if the secret overlaps v or v', but starts before it). We have $\min N, \min N' \geq -m$.

- If $N = N' = \emptyset$ then replacing $S_{\Sigma,k,j}$ with items from the fooling set trivially preserve equality of v and v'.
- Let $n = \min(N \cup N')$ and let $n' = \min(N')$ if $N \in I$ otherwise $n' = \min(N)$.
  - If n' does not exist or $n' \geq n + |L(S_{\Sigma,k,j})|$ then the left side of $S_{\Sigma,k,j}$ is a plaintext provided by attacker or secret other than $S_{\Sigma,k,j}$. By Claim 1, this happens with probability $\leq \frac{1}{|F^Q_{\Sigma,k,j}|}$.
  - Otherwise $L_2(S_{\Sigma,k,j})$ is a prefix of $vL_2(S_{\Sigma,k,j})$, where v is a plaintext provided by attacker or secret other than $S_{\Sigma,k,j}$. By Claim 2 (with $x = L_2(S_{\Sigma,k,j})$), we see that $L_2(S_{\Sigma,k,j})$ depends only on v. Again, by Claim 1, this happens with probability $\leq \frac{1}{|F^Q_{\Sigma,k,j}|}$.

4.5 Wrapping up

Let $P_{\texttt{bad}}$ be a probability that some secret is a bad token (it is limited by Lemma 3). Attacker making q queries to one oracle decreases fooling set size by q (with probability $1 - \frac{q}{|F_{\Sigma,k,j}|-q}$), and his probability of finding a bad token is $\leq 1 - qP_{\texttt{bad}}$. Therefore, after making q queries, the probability of success is $\geq 1 - \frac{q}{|F_{\Sigma,k,j}|-q} - qP_{\texttt{bad}}$. Obviously, making queries about more than one stream does not decrease this bound.

For parameters $C_2 = 10, \texttt{SECRETLEN} = 150, \texttt{MIN}\Sigma = 16, |\texttt{DICT}| < 10000$, we get the initial fooling set size $\approx 2^{41}$ and probability that a token is good $\approx 1 - 2^{-41}$ (and these values depend exponentially on C_2 and `SECRETLEN`). These bounds could be improved a lot by making bound in Lemma 2 tighter.

Therefore, for these parameters SafeDeflate is $(2^{25}, 2^{-15})$-*secure*.

5 Typical compression ratios

The example implementation together with tests of SafeDeflate is available at https://github.com/zielmicha/safedeflate.

We have downloaded several popular websites and concatenated them in two files: training data and test data. Most frequent 10-grams from training data were used to generate the dictionary. The following table compares sizes of the test dataset compressed using different algorithms.

Compression	Size
None	1772 KiB
Huffman coding only	1128 KiB
SafeDeflate $\Sigma_C =$ `byte`	876 KiB
SafeDeflate*[5] $\Sigma_C =$ `byte`	800 KiB
SafeDeflate $\Sigma_C =$ `alphanum`, `DICT` $= \emptyset$	588 KiB
SafeDeflate $\Sigma_C =$ `alphanum`	580 KiB
SafeDeflate $\Sigma_C =$ `hex`, `DICT` $= \emptyset$	432 KiB
Unsafe Deflate	296 KiB

It turns out that SafeDeflate variant optimized for compressing HTML is significally better than pure Huffman encoding and worse than unsafe deflate. If we restrict secret alphabet to alphanumeric or hexadecimal characters, the compression ratio becomes more favorable and dictionary becomes unnecessary. However the dictionary improves compression when we do not have any restrictions on secret alphabet.

2 Conclusion

In order to analyze CRIME-like attack, we have introduced a model called secrets in chosen container attack. We have designed a variant of Deflate, called SafeDeflate, and proven it secure in this model. SafeDeflate could be deployed on HTTP and TLS servers without any modifications to the clients.

[5] SafeDeflate* is a SafeDeflate variant with reduced security

References

[1] J. Rizzo and T. Doung, “The CRIME Attack.”

[2] “BREACH ATTACK” [Online]. Available: http://breachattack.com/

[3] “Defending against the BREACH attack” [Online]. Available: https://community.qualys.com/blogs/securitylabs/2013/08/07/defending-against-the-breach-attack

[4] J. Kelsey, “Compression and Information Leakage of Plaintext.”

[5] J. Czyzowicz and A. Pelc, “Solution of Ulam’s problem on binary search with two lies.”

[6] P. Deutsch, “DEFLATE Compressed Data Format Specification (RFC-1951).”