# Protecting Encrypted Cookies from Compression Side-Channel Attacks

Janaka Alawatugoda[1], Douglas Stebila[1,2], and Colin Boyd[3]

[1] School of Electrical Engineering and Computer Science,
[2] School of Mathematical Sciences,
Queensland University of Technology, Brisbane, Australia
janaka.alawatugoda@qut.edu.au, stebila@qut.edu.au
[3] Department of Telematics,
Norwegian University of Science and Technology, Trondheim, Norway
colin.boyd@item.ntnu.no

**Abstract.** Compression is desirable for network applications as it saves bandwidth; however, when data is compressed before being encrypted, the amount of compression leaks information about the amount of redundancy in the plaintext. This side channel has led to successful CRIME and BREACH attacks on web traffic protected by the Transport Layer Security (TLS) protocol. The general guidance in light of these attacks has been to disable compression, preserving confidentiality but sacrificing bandwidth. In this paper, we examine two techniques—heuristic separation of secrets and fixed-dictionary compression—for enabling compression while protecting high-value secrets, such as cookies, from attack. We model the security offered by these techniques and report on the amount of compressibility that they can achieve.

## 1 Introduction

To save communication costs, network applications often compress data before transmitting it; for example, the Hypertext Transport Protocol (HTTP) [1, §4.2] has an optional mechanism in which a server compresses the body of an HTTP response, most commonly using the gzip algorithm. When encryption is used to protect communication, compression must be applied before encryption (since ciphertexts should look random, they should have little apparent redundancy that can be compressed). In fact, to facilitate this, the Transport Layer Security (TLS) protocol [2, §6.2.2] has an optional compression mode that will compress all application data before encrypting it.

While compression is useful for reducing the size of transmitted data, it has had a negative impact when combined with encryption, because the amount of compression acts as a *side channel*. Most research considers side-channels such as timing [3,4] or power consumption [5], which can reveal information about cryptographic operations and secret parameters.

*Compression-based leakage.* In 2002, Kelsey [6] showed how compression can act as a form of side-channel leakage. If plaintext data is compressed before being encrypted, the length of the ciphertext reveals information about the amount of compression, which in turn can reveal information about the plaintext. Kelsey notes that this side channel differs from other types of side channels in two key ways: "it reveals information about the plaintext, rather than key material", and "it is a property of the algorithm, not the implementation".

Kelsey's most powerful attack is an *adaptive chosen input attack*: if an attacker is allowed to choose inputs $x$ that are combined with a target secret $s$ and the concatenation $x\|s$ is compressed and encrypted, observing the length of the outputs can eventually allow the attacker to extract the secret $s$. For example, to determine the first character of $s$, the attacker could ask to have the string $x =$ `prefix*prefix` combined with $s$, then compressed and encrypted, for every possible character `*`; in one case, when `*` $= s_1$, the amount of redundancy is higher and the ciphertext should be shorter. Once each character of $s$ is found, the attack can be carried out on the next character. The attack is somewhat noisy, but succeeds reasonably often.

Key to this attack is the fact that most compression algorithms (such as the DEFLATE algorithm underlying gzip) are *adaptive*: they adaptively build and maintain a *dictionary* of recently observed strings, and replace subsequent occurrences of that string with a code.

*The CRIME and BREACH attacks.* In 2012, Rizzo and Duong [7] showed how to apply Kelsey's adaptive chosen input attack against gzip compression as used in TLS, in what they called the *Compression Ratio Info-leak Mass Exploitation (CRIME)* attack. The primary target of the CRIME attack was the user's cookie in the HTTP header. If the victim visited an attacker-controlled web page, the attacker could use Javascript to cause the victim to send HTTP requests to URLs of the attacker's choice on a specified server. The attacker could adaptively choose those URLs to include a prefix to carry out Kelsey's adaptive chosen input attack. Some care is required to ensure the padding does not hide the length with block ciphers, but this can be dealt with. The CRIME attack also applies to compression as used in the SPDY protocol [8].

As a result of the CRIME attack, it was recommended that TLS compression be disabled, and the Trustworthy Internet Movement's SSL Pulse report for December 2014 finds that just 7.2% of websites have TLS compression enabled [9]; moreover, all major browsers have disabled it.

However, compression is also built into the HTTP protocol: servers can optionally compress the body of HTTP responses. While this excludes the cookie in the header, this attack can still succeed against secret values in the HTTP body, such as anti-cross-site request forgery (CSRF) tokens. Suggested by Rizzo and Duong, this was demonstrated by Gluck et al. [10] in the *Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext (BREACH)* attack.

*Mitigation techniques.* Gluck et al. [10] discussed several possible mitigation techniques against the BREACH attack, listed in decreasing order of effectiveness:

1. Disabling HTTP compression
2. Separating secrets from user input
3. Randomizing secrets per request
4. Masking secrets (effectively randomizing by XORing with a random nonce)
5. Length hiding (by adding a random number of bytes to the responses)
6. Rate-limiting the requests

Despite the demonstrated practicality of the BREACH attack, support for and use of HTTP compression remains widespread, due in large part to the value of decreasing communication costs and time. In fact, compression is even more tightly integrated into the proposed HTTP version 2 [11] than previous versions. Techniques 2–4 generally require changes to both browsers and web servers. For example, masking secrets such as anti-CSRF tokens requires new mark-up for secrets, which browsers and servers can interpret to apply the randomized masking technique. Techniques 5–6 can be unilaterally applied by web servers, though length hiding can be defeated with statistical averaging, and rate-limiting must find a balance between legitimate requests and information leakage.

*Related work.* There has been little academic study of compression and encryption. Besides Kelsey's adaptive chosen input attack and the related CRIME and BREACH attacks, the only relevant work we are aware of is that of Kelley and Tamassia [12]. They give a new security notion called *entropy-restricted semantic security* (ER-IND-CPA) for *keyed compression functions* which combine both encryption and compression: compared with the normal indistinguishability under chosen plaintext attack (IND-CPA) security notion, in ER-IND-CPA the adversary should not be able to distinguish between the encryption of two messages that *compress* to the same length. Kelley and Tamassia then show how to construct a cipher based on the LZW compression algorithm by rerandomizing the compression dictionary. Unfortunately, the ER-IND-CPA notion does not capture the CRIME and BREACH attacks, which depend on observing messages that compress to different lengths.

In leakage-resilient security definitions [13,14,15,16], leakage of the secret key is addressed. This differs from the setting in compression-based side-channel attacks, which addresses leakage of the plaintext. Thus, previous leakage-resilient approaches are not suitable to model compression-based side-channel attacks.

*Our contributions.* In this work, we study symmetric-key compression-encryption schemes, characterizing the security properties that can be achieved by various mitigation techniques in the face of CRIME- and BREACH-like attacks.

To some extent, the side channel exposed by compression is fundamentally unavoidable: if transmission of data is decreased, nothing can hide the fact that some redundancy existed in the plaintext. Hence, we focus our study on the ability of the attacker to learn specific "high value" secrets embedded in a plaintext, such as cookies or anti-CSRF tokens. In our models, we imagine there is a secret value $ck$, and the adversary can adaptively obtain encryptions

$$\mathrm{Enc}_k(m' \| ck \| m'') \tag{1}$$

for prefix $m'$ and suffix $m''$ of its choice; the attacker's goal is to learn about $ck$.

The first mitigation technique we consider is that of *separating secrets*. During compression/encryption, an application-aware filter is applied to the plaintext to separate out any potential secret values from the data, the remaining plaintext is compressed, then the secrets and compressed plaintext are encrypted; after decryption, the inverse of the filter is used to reinsert the secret values in the decompressed plaintext. Assuming the filter fully separates out all secret values, we show that the separating secrets technique is able to achieve a strong notion of protection, which we call *chosen cookie indistinguishability* (CCI): the adversary cannot determine which of two cookies $ck_0$ and $ck_1$ of the adversary's choice was encrypted with messages of the adversary's choice given ciphertexts as in (1).

The second mitigation technique we consider is the use of a *fixed-dictionary compression scheme*, where the dictionary used for compression does not adapt to the plaintext being compressed, but instead is preselected in advance based on the expected distribution of plaintext messages, for example including common English words like "the" and "and". We show that, if the secret values are sufficiently high entropy, then fixed-dictionary compression is able to achieve *cookie recovery* (CR) security: if the secret cookie is chosen uniformly at random, the adversary cannot recover the entire secret cookie even given an adaptive message attack as in (1). While cookie recovery security does not meet the "gold standard" of indistinguishability notions for encryption, it may be sufficient for some settings, for example protecting compressed HTTP traffic from CRIME and BREACH attacks that try to recover cookies and anti-CSRF tokens.

We also characterize the relationship among the CCI and CR security notions, as well as an intermediate notion called *random cookie indistinguishability* (RCI) and the ER-IND-CPA notion of Kelley and Tamassia [12].

In the separating secrets technique, if the number of secrets extracted by the separating filter is relatively small, then the compressibility generally remains close to that of normal compression of the full plaintext. In the fixed-dictionary compression technique, compressibility suffers quite a bit compared to adaptive techniques on the full plaintext, although if the dictionary is constructed from a corpus of text similar to the plaintext, then some compression can be achieved.

Figure 1 summarizes experimental results comparing compression ratios for these two techniques on the HTML, CSS, and Javascript source code of the top 10 global websites as reported by Alexa Top Sites (`http://www.alexa.com/topsites`). On average, the compression ratio (uncompressed : compressed size) of gzip applied to the full source code was 5.42×; applying a separation filter that extracted all values following `value=` in the HTML source code yielded an average compression ratio of 5.20×; compression of each page using a fixed dictionary trained on all 10 pages yielded an average compression ratio of 1.55×.

## 2 Definitions

*Notation.* If $x$ is a string, then $x_i$ denotes the $i$th character of $x$; $x_{i:\ell}$ denotes the length-$\ell$ substring of $x$ starting at position $i$: $x_{i:\ell} = x_i \| \ldots \| x_{i+\ell-1}$. If $x$ and $y$

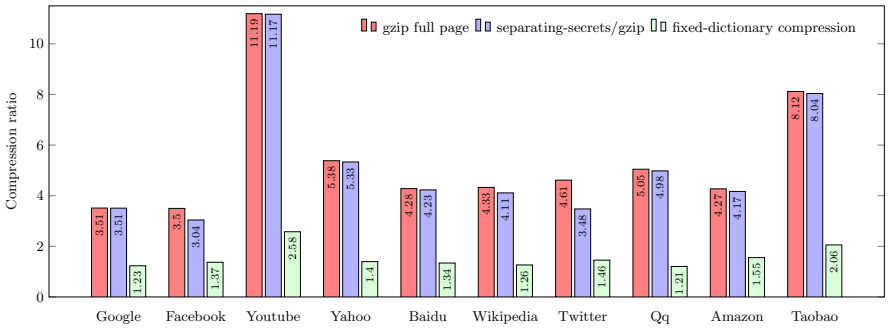


**Fig. 1.** Compression ratios of full page compression versus mitigation techniques

are strings, then $x \preceq y$ denotes that $x$ is a substring of $y$. The *index* of $x$ in $y$ is the smallest $i$ such that $y_{i:|x|} = x$ and is denoted by $\text{ind}_y(x)$; if $x \not\preceq y$, we denote $\text{ind}_y(x) = \perp$. The empty string is denoted by $\epsilon$.

### 2.1 Encryption and compression schemes

Recall the standard definition of an encryption scheme:

**Definition 1 (Symmetric-key encryption).** *A* symmetric-key encryption scheme *$\Pi$ for message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ is a tuple of algorithms:*

- $\text{KeyGen}() \xrightarrow{\$} k$*: A probabilistic* key generation algorithm *that generates a random key $k$ in the keyspace $\mathcal{K}$.*
- $\text{Enc}_k(m) \xrightarrow{\$} c$*: A possibly probabilistic* encryption algorithm *that takes as input a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$ and outputs a ciphertext $c \in \mathcal{C}$.*
- $\text{Dec}_k(c) \to m'$ *or* $\perp$*: A deterministic* decryption algorithm *that takes as input a key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$, and outputs either a message $m' \in \mathcal{M}$ or an error symbol $\perp$.*

Correctness of symmetric-key encryption is defined in the obvious way: for all $k \xleftarrow{\$} \text{KeyGen}()$ and all $m \in \mathcal{M}$, we require that $\text{Dec}_k(\text{Enc}_k(m)) = m$.

**Definition 2 (Compression scheme).** *A* compression scheme *$\Gamma$ for message space $\mathcal{M}$ with output space $\mathcal{O}$ is a pair of algorithms:*

- $\text{Comp}(m) \xrightarrow{\$} o$*: A possibly probabilistic* compression *algorithm that takes as input a message $m \in \mathcal{M}$ and outputs an encoded value $o \in \mathcal{O}$.*
- $\text{Decomp}(o) \to m'$ *or* $\perp$*: A* decompression *algorithm that takes as input an encoded value $o \in \mathcal{O}$ and outputs a message $m' \in \mathcal{M}$ or an error symbol $\perp$.*

Note that $|\text{Comp}(m)|$ may not necessarily be less than $|m|$; Shannon's coding theorem implies that no algorithm can encode every message with shorter length, so not all messages may actually be "compressed": some may increase in lenth.

Correctness of a compression scheme is again defined in the obvious way: for all $m \in \mathcal{M}$, we require that $\text{Decomp}(\text{Comp}(m)) = m$.

In this paper, we are interested in *symmetric-key compression-encryption schemes*, which formally are just symmetric-key encryption schemes as in Definition 1, but usually have the goal of outputting shorter ciphertexts via some form of compression. Of course, every symmetric-key encryption scheme is also a symmetric-key compression-encryption scheme, with "compression" being the identity function. We will often deal with the following specific, natural composition of compression and symmetric-key encryption:

**Definition 3 (Composition of compression and encryption).** *Let $\Gamma = (\text{Comp}, \text{Decomp})$ be a compression scheme with message space $\mathcal{M}$ and output space $\mathcal{O}$. Let $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ be a symmetric-key encryption scheme with message space $\mathcal{O}$ and ciphertext space $\mathcal{C}$. The symmetric-key compression-encryption scheme $\Pi \circ \Gamma$ constructed from $\Gamma$ and $\Pi$ is the following tuple:*

$$\begin{aligned}(\Pi \circ \Gamma).\text{KeyGen}() &= \Pi.\text{KeyGen}() \\ (\Pi \circ \Gamma).\text{Enc}_k(m) &= \Pi.\text{Enc}_k(\Gamma.\text{Comp}(m)) \\ (\Pi \circ \Gamma).\text{Dec}_k(c) &= \Gamma.\text{Decomp}(\Pi.\text{Dec}_k(c))\end{aligned}$$

Note that $\Pi \circ \Gamma$ is itself a symmetric-key encryption scheme with message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. If $\Gamma$ and $\Pi$ are both correct, then so is $\Pi \circ \Gamma$.

### 2.2 Existing security notions

The standard security notion for symmetric-key encryption is indistinguishability of encrypted messages. In this paper, we focus on chosen plaintext attack. The security experiment $\text{Exp}^{\mathsf{IND\text{-}CPA}}_{\Pi}(\mathcal{A})$ for indistinguishability under chosen plaintext attack (IND-CPA) of a symmetric-key encryption scheme $\Pi$ against a stateful adversary $\mathcal{A}$ is given in Figure 2. The *advantage* of $\mathcal{A}$ in breaking the IND-CPA experiment for $\Pi$ is $\text{Adv}^{\mathsf{IND\text{-}CPA}}_{\Pi}(\mathcal{A}) = \left|2\Pr\left(\text{Exp}^{\mathsf{IND\text{-}CPA}}_{\Pi}(\mathcal{A}) = 1\right) - 1\right|$.

Kelley and Tamassia [12] give a definition of *entropy-restricted* IND-CPA *security* which applies to keyed compression schemes $\Pi$, and demands indistinguishability of encryptions of messages from the same class $\mathcal{L} \subseteq \mathcal{M}$; typically, $\mathcal{L}$ is the class of messages that encrypt (compress) to the same length under $\Pi.\text{Enc}$, such as:

$$\mathcal{L}_\ell = \{m \in \mathcal{M} : |\Pi.\text{Enc}(m)| = \ell\} \quad .$$

The ER-IND-CPA security experiment is given in Figure 2; the corresponding advantage is defined similarly. Kelley and Tamassia note that any IND-CPA-secure symmetric-key encryption scheme $\Pi$, combined with any compression scheme $\Gamma$, is immediately ER-IND-CPA-secure. As well, it is easily seen that if a symmetric-key encryption scheme is ER-IND-CPA-secure for the class $\mathcal{L}_\ell = \{m \in \mathcal{M} : |m| = \ell\}$, then that scheme is also an IND-CPA-secure symmetric-key encryption.

$\underline{\mathrm{Exp}^{\mathsf{IND\text{-}CPA}}_{\Pi}(\mathcal{A})}$

$k \xleftarrow{\$} \Pi.\mathrm{KeyGen}()$
$b \xleftarrow{\$} \{0,1\}$
$(m_0, m_1, st) \xleftarrow{\$} \mathcal{A}^{E}()$
**if** $|m_0| \neq |m_1|$, **then return** $\bot$
$c \leftarrow \Pi.\mathrm{Enc}_k(m_b)$
$b' \xleftarrow{\$} \mathcal{A}^{E}(c, st)$
**return** $(b' = b)$

$\underline{E(m)}$

**return** $\Pi.\mathrm{Enc}_k(m)$

$\underline{\mathrm{Exp}^{\mathsf{ER\text{-}IND\text{-}CPA}}_{\Pi,\mathcal{L}}(\mathcal{A})}$

$k \xleftarrow{\$} \Pi.\mathrm{KeyGen}()$
$b \xleftarrow{\$} \{0,1\}$
$(m_0, m_1, st) \xleftarrow{\$} \mathcal{A}^{E}()$
**if** $m_0 \notin \mathcal{L}$ or $m_1 \notin \mathcal{L}$, **then return** $\bot$
$c \leftarrow \Pi.\mathrm{Enc}_k(m_b)$
$b' \xleftarrow{\$} \mathcal{A}^{E}(c, st)$
**return** $(b' = b)$

$\underline{E(m)}$

**return** $\Pi.\mathrm{Enc}_k(m)$

**Fig. 2.** Security experiments for indistinguishability under chosen plaintext attack (IND-CPA, left) and entropy-restricted IND-CPA (ER-IND-CPA, right)

### 2.3 New security notions

In this paper, we focus on the ability of an attacker to learn about a secret piece of data inside a larger piece of data, where the attacker controls everything except the secret data. We use the term *cookie* to refer to the secret data; in practice, this could be an HTTP cookie in a header, an anti-CSRF token, or some piece of personal information. We will allow the attacker to adaptively obtain encryptions of compressions of data of the form $m' \| ck \| m''$ for a secret cookie $ck$ and adversary-chosen message prefix $m'$ and suffix $m''$.

We now present three notions for the security of cookies in the context of compression-encryption schemes:

- *Cookie recovery* (CR) *security*: A simple, but relatively weak, security notion for symmetric-key compression-encryption schemes: it should be hard for the attacker to *fully recover* a secret value, even given adaptive access to an oracle that encrypts plaintexts of its choosing with the target cookie embedded.
- *Random cookie indistinguishability* (RCI) *security*: The adversary has to decide which of two randomly chosen cookies was embedded in the encrypted plaintext, given adaptive access to an oracle that encrypts plaintexts of its choosing with the target cookie embedded.
- *Chosen cookie indistinguishability* (CCI) *security*: Here, the adversary has to decide which of two cookies of the adversary's choice was embedded in the encrypted plaintext, given adaptive access to an oracle that encrypts plaintexts of its choosing with the target cookie embedded.

These security notions are formalized in the following definition, which refers to the security experiments shown in Figure 3.

**Definition 4 (CR, RCI, CCI security).** *Let $\Psi$ be a symmetric-key compression-encryption scheme. Let $\mathcal{A}$ denote an algorithm. Let $\mathcal{CK}$ denote the cookie space. Let* $\mathsf{xxx} \in \{\mathsf{CR}, \mathsf{RCI}, \mathsf{CCI}\}$ *be a security notion. Consider the security experiment*

$\underline{\mathrm{Exp}^{\mathsf{CR}}_{\Psi,\mathcal{CK}}(\mathcal{A})}$

1: $k \xleftarrow{\$} \Psi.\mathrm{KeyGen}()$
2: $ck \xleftarrow{\$} \mathcal{CK}$
3: $ck' \xleftarrow{\$} \mathcal{A}^{E_1,E_2}()$
4: **return** $(ck' = ck)$

$\underline{E_1(m', m'')}$

1: **return** $\Psi.\mathrm{Enc}_k(m'\|ck\|m'')$

$\underline{E_2(m)}$

1: **return** $\Psi.\mathrm{Enc}_k(m)$

$\underline{\mathrm{Exp}^{\mathsf{RCI/CCI}}_{\Psi,\mathcal{CK}}(\mathcal{A})}$

1: $k \xleftarrow{\$} \Psi.\mathrm{KeyGen}()$
2: **if** RCI **then**
3: $\quad (ck_0, ck_1) \xleftarrow{\$} \mathcal{CK}$ s.t. $|ck_0| = |ck_1|$; $st \leftarrow \perp$
4: **else if** CCI **then**
5: $\quad (ck_0, ck_1, st) \xleftarrow{\$} \mathcal{A}^{E_2}()$ s.t. $|ck_0| = |ck_1|$
6: $b \xleftarrow{\$} \{0, 1\}$
7: $b' \xleftarrow{\$} \mathcal{A}^{E_1,E_2}(ck_0, ck_1, st)$
8: **return** $(b' = b)$

$\underline{E_1(m', m'')}$

1: **return** $\Psi.\mathrm{Enc}_k(m'\|ck_b\|m'')$

$\underline{E_2(m)}$

1: **return** $\Psi.\mathrm{Enc}_k(m)$

**Fig. 3.** Security experiments for cookie recovery (left) and random cookie indistinguishability and chosen cookie indistinguishability (right) attacks

*$\mathrm{Exp}^{\mathsf{xxx}}_{\Psi,\mathcal{CK}}(\mathcal{A})$ in Figure 3. Define $\mathrm{Adv}^{\mathsf{CR}}_{\Psi,\mathcal{CK}}(\mathcal{A}) = \Pr\left(\mathrm{Exp}^{\mathsf{CR}}_{\Psi,\mathcal{CK}}(\mathcal{A}) = 1\right)$ as the probability that $\mathcal{A}$ wins the cookie recovery experiment for $\Psi$ and $\mathcal{CK}$. Similarly, define $\mathrm{Adv}^{\mathsf{xxx}}_{\Psi,\mathcal{CK}}(\mathcal{A}) = \left|2\Pr\left(\mathrm{Exp}^{\mathsf{xxx}}_{\Psi,\mathcal{CK}}(\mathcal{A}) = 1\right) - 1\right|$, $\mathsf{xxx} \in \{\mathsf{RCI}, \mathsf{CCI}\}$, as the advantage that $\mathcal{A}$ has in winning the random cookie and chosen cookie indistinguishability experiments.*

*Remark 1.* The CR, RCI, and CCI security notions intentionally include only the confidentiality of the cookie as a security goal, and not the confidentiality of any non-cookie data in the rest of the message. In most applications it would be desirable to obtain confidentiality of non-cookie data as well, and in many real-world situations, the application layer's cookie and non-cookie data are jointly sent to the security layer (such as SSL/TLS) for encryption. Our notions do not preclude the scheme from encrypting the non-cookie data as well (and in fact our constructions in Sections 3 and 4 do so). However, it is not possible in general to require confidentiality of the non-cookie data while still allowing it to be compressed, as that brings us back around to the original problem that motivated the work—compression of adversary-provided data can lead to ciphertexts of different lengths that break indistinguishability. This cycle can be broken by demanding some length restriction on the separated non-cookie data, such as in the ER-IND-CPA notion described in Section 2.2, but we omit that complication to focus solely on the security of the high-value secret cookies.

### 2.4 Relations and separations between security notions

Cookie recovery, being a computational problem rather than a decisional problem, is a weaker security notion. Keeping CR as an initial step, the RCI and CCI notions gradually increase the security afforded to the cookie.

The following relations exist between security notions for symmetric-key compression-encryption schemes:

$$\mathsf{CCI} \implies \mathsf{RCI} \implies \mathsf{CR} \ .$$

In other words, every scheme that provides chosen cookie indistinguishability provides random cookie indistinguishability, and so on. Moreover, these notions are distinct, and we can show separations between them:

$$\mathsf{CR} \not\Longrightarrow \mathsf{RCI} \not\Longrightarrow \mathsf{CCI} \ .$$

Additionally, we can connect our new notions with existing notions:

$$\mathsf{ER\text{-}IND\text{-}CPA} \implies \mathsf{IND\text{-}CPA} \implies \mathsf{CCI} \qquad \text{and} \qquad \mathsf{CCI} \not\Longrightarrow \mathsf{IND\text{-}CPA} \ .$$

These last relations should be interpreted as follows. A standard (non-compressing) IND-CPA-secure symmetric-key encryption scheme is also CCI-secure. This is not to say, however, that an IND-CPA-secure symmetric-key encryption scheme combined with a compression scheme, such as $\Pi \circ \Gamma$ in Definition 3, is CCI-secure.

The proofs of these relations and counterexamples for the separations appear in the full version of the paper [17]. A brief discussion of the intuition for each appears in Appendix A.

## 3 Technique 1: Separating secrets from user inputs

In this section we analyze a mitigation technique against attacks that recover secrets from compressed data: separating secrets from user inputs. The basic idea of separating secrets from user inputs is: given an input, use a filter to separate all the secrets from the rest of the content, including user inputs. Then the rest of the content is compressed, while the secrets are kept uncompressed. This mitigation technique is a generic mitigation technique against a whole class of compression-based side-channel attacks.

### 3.1 The scheme

**Definition 5 (Filter).** *A* filter *is an invertible (efficient) function* $f : \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$.

Given a filter $f$ and a compression scheme $\Gamma$, the separating-secrets scheme $\mathrm{SS}_{f,\Gamma}$ is given in Figure 4.

Our results will make use of the following two conditions on filters. Intuitively, a filter is *effective* if it removes cookies from an input string, and is *safe* if no prefix/suffix can fool the filter into separating out one cookie but not another.

**Definition 6 (Effective filter).** *Let $\mathcal{CK}$ be a cookie space, and let $f$ be a filter. We say that $f$ is* effective at separating out $\mathcal{CK}$ *if, for all $ck \in \mathcal{CK}$ and all $m', m''$, we have that $ck \not\preceq y$, where $(x, y) = f(m' \| ck \| m'')$.*

$\underline{\mathrm{SS}_{f,\Gamma}.\mathrm{Comp}(m)}$

1: $(pt_s, pt_{ns}) \leftarrow f(m)$
2: $\widetilde{pt_{ns}} \leftarrow \Gamma.\mathrm{Comp}(pt_{ns})$
3: **return** $pt_s \| \widetilde{pt_{ns}}$

$\underline{\mathrm{SS}_{f,\Gamma}.\mathrm{Decomp}(pt)}$

1: Parse $pt_s \| \widetilde{pt_{ns}} \leftarrow pt$
2: $pt_{ns} \leftarrow \Gamma.\mathrm{Decomp}(\widetilde{pt_{ns}})$
3: $m \leftarrow f^{-1}(pt_s, pt_{ns})$
4: **return** $m$

**Fig. 4.** Abstract separating-secrets compression scheme SS

**Definition 7 (Safe filter).** *Let $\mathcal{CK}$ be a cookie space, and let $f$ be a filter. We say that $f$ is* safe for $\mathcal{CK}$ *if, for all $ck_0, ck_1 \in \mathcal{CK}$ such that $|ck_0| = |ck_1|$ and all $m', m''$, we have that $|x_0| = |x_1|$ and $y_0 = y_1$, where $(x_0, y_0) = f(m' \| ck_0 \| m'')$ and $(x_1, y_1) = f(m' \| ck_1 \| m'')$.*

*Example cookie space and filter.* Let $\lambda \in \mathbb{N}$ and let $\mathcal{CK}$ be the set of alphanumeric strings starting with the literal "`secret`" and starting and ending with a space (denoted by $\llcorner$), i.e., strings matched by the regular expression

$$\llcorner\texttt{secret[A-Za-z0-9]}^{\lambda}\llcorner$$

Let $f$ be a filter that uses the above regular expression to separate out secrets. Consider a string of the form $m = m_0 \llcorner ck_1 \llcorner m_1 \llcorner ck_2 \llcorner m_2 \llcorner \ldots \llcorner ck_n \llcorner m_n$, where $m_i$ contains no substring matching the above regular expression and $ck_i$ is a string completely matching the above regular expression (excluding the initial and terminal space $\llcorner$). Then $f(m) = (pt_s, pt_{ns})$, where $pt_s = ck_1 \| \ldots \| ck_n$ and $pt_{ns} = m_0 \| \tau \| m_1 \| \tau \| \ldots \| m_n$, and $\tau$ represents a fixed replacement token that can not appear as a substring of any $m \in \mathcal{M}$. The above filter $f$ is effective at separating out and safe for the above $\mathcal{CK}$. The intuitive reason for this is that, since each cookie begins and ends with a character $\llcorner$ which does not appear within the cookie, no prefix or suffix can cause the filter to not separate a cookie.

### 3.2 CCI security of basic separating-secrets technique

In this section we analyze the security of separating-secrets mitigation technique according to CCI notion. Let $\Pi = (\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ be an IND-CPA-secure symmetric-key encryption scheme and $\mathrm{SS}_{f,\Gamma}$ be the separating-secrets compression scheme given in Figure 4. We consider the security of the resulting symmetric-key compression-encryption scheme $\Pi \circ \mathrm{SS}_{f,\Gamma}$, showing that, if the filter $f$ safely separates out cookies, then breaking chosen cookie indistinguishability of $\Pi \circ \mathrm{SS}_{f,\Gamma}$ is as hard as breaking indistinguishability (IND-CPA) of encryption scheme $\Pi$. The proof of Theorem 1 appears in Appendix B.

**Theorem 1.** *Let $\Pi$ be a symmetric-key encryption scheme and let $\Gamma$ be a compression scheme. Let $\mathcal{CK}$ be a cookie space, and let $f$ be a filter that is safe for $\mathcal{CK}$. Let $\mathcal{A}$ be any adversary against the* CCI *security of the separating-secrets symmetric-key compression-encryption scheme $\Pi \circ \mathrm{SS}_{f,\Gamma}$, and let $q$ denote the number of queries that $\mathcal{A}$ makes to its $E_1$ oracle. Then* $\mathrm{Adv}^{\mathsf{CCI}}_{\Pi \circ \mathrm{SS}_{f,\Gamma}, \mathcal{CK}}(\mathcal{A}) \leq$

$q \cdot \mathrm{Adv}_{\Pi}^{\mathsf{IND\text{-}CPA}}(\mathcal{B}^{\mathcal{A}})$ , *where* $\mathcal{B}$ *is an algorithm, constructed using the adversary* $\mathcal{A}$ *as described in the proof, against the* IND-CPA *security of the symmetric-key encryption scheme* $\Pi$*.*

### 3.3 Separating secrets in HTML

Separating secrets from user inputs is a realistic mitigation technique against the BREACH attack: in the application layer, some fields which contain secrets (such as anti-CSRF tokens) can be identified and separated from the HTTP response body. In order to implement separating secrets from user inputs in HTML we need to describe a filter $f_{\mathrm{HTML}}$.

One possible method to separate secrets in HTML is to separate the content assigned to the `value` attribute of HTML elements. Among other uses, the `value` attribute defines the value of a specific field in a form. The HTML code segment of Figure 5 shows inclusion of a secret anti-CSRF token as a hidden `input` field in a web form, which will appear in a HTML response body. By separating the content in the `value` attribute, we separate the anti-CSRF token.

```
<form action="/money_transfer" method="post">
<input type="hidden" name="csrftoken"
       value="OWT4NmQlODE4ODRjN2Q1NTlhMmZlYWE...">
...
</form>
```

**Fig. 5.** HTML code segment showing inclusion of anti-CSRF token in a web form

The following (case-insensitive) regular expression can be used to separate out quoted anti-CSRF tokens in the `value` attribute of HTML elements:

```
value\s*=\s*"[A-Za-z0-9]+"|value\s*=\s*'[A-Za-z0-9]+'
```

This filter is effective at separating out and safe for the implied set of cookies, in the sense of Definitions 6 and 7.

However, the above regular expression is not perfect, highlighting the challenges of using heuristic techniques to separate out secrets.

First, the above regular expression will also capture the `value` attribute of HTML elements other than hidden `input` elements, such as `option`, which may not need to be treated as secret, so it is not as efficient as it could be.

Second, the above regular expression does not capture anti-CSRF tokens in unquoted `value` attributes, such as `value=OWT4NmQl`, which are allowed by the HTML specification. While it is easy to add an additional term such as `|value\s*=\s*[A-Za-z0-9]+` to the regular expression to capture unquoted attributes, this filter would no longer be effective in the sense of Definition 6: if a cookie is `value=OWT4NmQl`, and the adversary constructs $m' =$ `value=`, then $m' \| ck =$ `value=value=OWT4NmQl`, and the filter applied to $m' \| ck$ would separate out `value=value` as the cookie and leave `=OWT4NmQl` unprotected.

**Table 1.** Compression performance (file size in bytes and compression ratio) for separating secrets (Section 3) and fixed dictionary (Section 4) techniques

| Website | Uncompressed | gzip full page | Separating secrets | Fixed dictionary |
|---|---|---|---|---|
| Google.com | 145 599 | 41 455 (3.51×) | 41 502 (3.51×) | 117 794 (1.23×) |
| Facebook.com | 48 226 | 13 785 (3.50×) | 15 863 (3.04×) | 35 036 (1.37×) |
| Youtube.com | 467 928 | 41 813 (11.19×) | 41 893 (11.17×) | 181 676 (2.58×) |
| Yahoo.com | 444 408 | 82 572 (5.38×) | 83 342 (5.33×) | 318 386 (1.40×) |
| Baidu.com | 74 979 | 17 519 (4.28×) | 17 727 (4.23×) | 55 950 (1.34×) |
| Wikipedia.org | 48 548 | 11 217 (4.33×) | 11 809 (4.11×) | 38 406 (1.26×) |
| Twitter.com | 57 777 | 12 520 (4.61×) | 16 618 (3.48×) | 39 712 (1.46×) |
| Qq.com | 626 297 | 124 108 (5.05×) | 125 747 (4.98×) | 519 830 (1.21×) |
| Amazon.com | 234 609 | 54 922 (4.27×) | 56 278 (4.17×) | 150 924 (1.55×) |
| Taobao.com | 192 068 | 23 658 (8.12×) | 23 898 (8.04×) | 93 410 (2.06×) |

### 3.4 Experimental results on separating-secrets in HTML

Table 1 shows the result of applying the above regular expression to separate secrets on the top 10 global websites of Alexa Top Sites. As most pages contain little data in `value` attributes, the total amount of space required to transmit the separated secrets plus the remaining data is not much more than when the full page is compressed. (Table 1 also contains performance results of the fixed dictionary technique, to be discussed in Section 4.)

### 3.5 Discussion

The main drawback of the separating secrets technique is that the separation filter must be application-dependent. We noted already the challenges in using the heuristic regular expression above to capture anti-CSRF tokens: it may separate out non-secrets as well as secrets (which yields suboptimal compression) and it does not capture unquoted tokens (which is a problem for security).

Moreover, this HTML filter also only captures secrets in a `value` attribute, which does not necessarily capture all values that might be considered sensitive. For example, should the titles of books in a search results page on an shopping site be considered secret? If so, an alternative separation filter would have to be developed. To provide complete certainty, secret separation would require additional markup with which the developer clearly identifies which data should be treated as secret. Otherwise, any sensitive values which are not separated may be compressed together with user inputs and other application data, and hence remain open to the compression-based side-channel.

## 4 Technique 2: Fixed-dictionary compression

The CRIME and BREACH attacks work because the dictionary constructed by the DEFLATE compression algorithm is adaptive: if the attacker injects a substring of the target secret into the plaintext nearby the secret itself, then the plaintext will compress more because of the repeated substring. Some early

compression algorithms were non-adaptive, using a fixed dictionary mechanism. For example, Pike [18] used a fixed dictionary of 205 popular English words and a variable length coding mechanism to compress typical English text at a rate of less than 4 bits per character. Another recent algorithm, Smaz [19], similarly uses a fixed dictionary consisting of common digrams and trigrams from English and HTML source code, allowing it to compress even very short strings. Because the CRIME and BREACH attacks rely on the adaptivity of the compression dictionary, fixed-dictionary algorithms can offer resistance to such attacks while still providing some compression, albeit not as good as adaptive compression.

In this section, we investigate the use of fixed-dictionary compression in the context of encryption. We describe the basic idea of fixed-dictionary compression. We show that fixed-dictionary compression-encryption schemes can satisfy cookie recovery security for sufficiently large cookies. We then present an example of a modern fixed-dictionary compression algorithm and report on the compression ratios achieved by our algorithm.

### 4.1 The scheme

In general, fixed-dictionary compression schemes work by advancing through the string $x$ and looking to see if the current substring appears in the dictionary $\mathcal{D}$: if it does, then an encoding of the index of the substring is recorded, otherwise an encoding of the current substring is recorded. The compression scheme must specify the encoding rules in a way that unambiguously discriminates between the two cases to allow for correct decompression.

An abstract version of a fixed-dictionary fixed-width compression algorithm FD is given in Figure 6. FD checks if the current substring of length $w$ appears in the dictionary $\mathcal{D}$. If it does, it records the index of the substring in $\mathcal{D}$ and advances $w$ characters. If it does not, it records the next $\ell$ characters directly, then advances. (Using $\ell > 1$ but $\ell < w$ may be more efficient when it comes to encodings.) One could treat $\mathcal{D}$ either as a set of strings (recording which element is matched) or a long string (recording the starting and ending position of the matching substring); we will use the latter in the rest of this section.

For example, if $\mathcal{D}$ ="`cookierecoveryattack`", then $\mathrm{FD}_{\mathcal{D},4,2}$.Comp("`recover the cookie`") yields `7ver_the_1ie`.

### 4.2 CR security of basic fixed-dictionary technique

Let $\Pi$ be a symmetric-key encryption scheme. Let $\mathcal{D}$ be a dictionary of length $d$ and $\mathrm{FD}_{\mathcal{D},w,\ell}$ be the abstract fixed-dictionary compression scheme in Figure 6.

Suppose the cookie space is binary strings of length $8\lambda$, or equivalently byte strings of length $\lambda$: $\mathcal{CK} = \{\texttt{0x00}, \ldots, \texttt{0xFF}\}^{\lambda}$.

If $\Pi$ is a secure encryption scheme, then, intuitively, the only way the adversary can learn information about the cookie from seeing ciphertexts $\mathrm{Enc}_k(\cdot \| ck \| \cdot)$ and $\mathrm{Enc}_k(\cdot)$ is from the length of the ciphertext: if some substring of $ck$ appears in the dictionary $\mathcal{D}$, then $ck$ will compress, and that length difference tells the adversary that the secret cookie is restricted to some subset of $\mathcal{CK}$ matching $\mathcal{D}$.

$\underline{\mathrm{FD}_{\mathcal{D},w,\ell}.\mathrm{Comp}(x)}$

1: $y \leftarrow \epsilon$
2: $i \leftarrow 1$
3: **while** $i \leq |x| - w + 1$ **do**
4: **if** $x_{i:w} \preceq \mathcal{D}$ **then**
5: $y \leftarrow y \parallel$ encoding of $\mathrm{ind}_{\mathcal{D}}(x_{i:w})$
6: $i \leftarrow i + w$
7: **else**
8: $y \leftarrow y \parallel$ encoding of $x_{i:\ell}$
9: $i \leftarrow i + \ell$
10: **return** $y$

$\underline{\mathrm{FD}_{\mathcal{D},w,\ell}.\mathrm{Decomp}(y)}$

1: $x \leftarrow \epsilon$
2: $i \leftarrow 1$
3: **while** $i \leq |y|$ **do**
4: **if** $y_i$ encodes an index **then**
5: $x \leftarrow x \parallel \mathcal{D}_{y_i:w}$
6: $i \leftarrow i + 1$
7: **else**
8: $x \leftarrow x \parallel$ decoding of $y_{i:\ell'}$
9: $i \leftarrow i + \ell'$
10: **return** $x$

**Fig. 6.** Abstract fixed-dictionary fixed-width compression scheme FD
Note the simplification that $\ell$ characters of $x$ are encoded as $\ell'$ characters of $y$.

The situation is subtler in the full CR experiment: the attacker can provide $m'$ and $m''$ and get $\mathrm{Enc}_k(\mathrm{Comp}(m'\|ck\|m''))$. If the last few bytes of $m'$ followed by the first few bytes of $ck$ appear in $\mathcal{D}$, then the string will compress more. This allows the attacker to carry out a CRIME-like attack on the first few bytes of $ck$.

For example, let $w = 4$ and suppose $\mathcal{D} =$ `1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ` and $\mathcal{CK} = [\texttt{0-9A-F}]^{\lambda}$. The attacker can query $m' =$ `890`, $m' =$ `90A`, $m' =$ `0AB`, .... In exactly one case, the adversary's $m'$ combined with the cookie's first byte will be in the dictionary, telling the adversary $ck_1$. For example, if $ck_1 =$ `B`, then when the adversary queries $m' =$ `90A`, the value that is compressed and then encrypted is $m'\|ck\|m'' =$ `90AB`..., which is a substring of $\mathcal{D}$.

While this allows the attacker to recover the first byte or two of the secret cookie with decent probability, it drops off exponentially; a similar argument applies to the last few bytes of the secret cookie. Theorem 2 captures this issue. Theorem 2 only provides quantifiable security when the cookie length $n$ is significantly bigger than the compression window $w$. Additionally, this type of attack on the first/last few bytes of the cookie precludes *indistinguishable* security, which is why we focus on cookie *recovery* here. (Admittedly, in some settings recovering the first/last few cookie bytes may still be quite damaging.)

**Theorem 2.** *Let $\Pi$ be a symmetric-key encryption scheme. Let $\mathcal{D}$ be a dictionary of $d$ words, each of length $\ell$. Let $w$ be positive integer. Let $\mathcal{CK} = \Omega^n$. Let $\mathcal{A}$ be any adversary against the cookie recovery security of the fixed-dictionary symmetric-key compression-encryption scheme $\Pi \circ \mathrm{FD}_{\mathcal{D},w,\ell}$. Then* $\mathrm{Adv}^{\mathsf{CR}}_{\Pi\circ\mathrm{FD}_{\mathcal{D},w,\ell}}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\Pi}(\mathcal{B}) + 2^{-\Delta}$ *, where $\mathcal{B}$ is an algorithm, constructed using adversary $\mathcal{A}$, against the* IND-CPA *security of the symmetric-key encryption scheme $\Pi$, and*

$$\Delta \geq \left(1 - d\left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-3w+1}\right)\right) \cdot \log_2\left(|\Omega|^{n-2w} - |\Omega|^{n-2w} \cdot d\left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-3w+1}\right)\right) .$$

For example, for cookies of $n = 16$ bytes, with a dictionary of $d = 4000$ words each of length $w = 4$, we have $\Delta \geq 63.999695$. Doubling $d$ gives $\Delta \geq 63.999391$.

The derivation of the formula in Theorem 2 appears in Appendix C.

### 4.3 Experimental results on fixed-dictionary technique

Table 1 shows the result of applying a fixed-dictionary based compression algorithm to the top 10 global websites of Alexa Top Sites. The 4000-byte dictionary was built from the most common 8-, 16-, and 32-character substrings of the pages. The compression algorithm was based in part on the Smaz [19] algorithm and was adapted slightly from Figure 6, to allow for variable-length words to be matched. Specifically, when attempting to encode the substring at the current position at line 4 in Figure 6, we first try variable length words in order of decreasing length, checking to see if $w = 18$, then $w = 16$, then $\ldots$, then $w = 4$ characters can be found in the dictionary. This requires the encoding to include both index and length of the dictionary substring.

- To encode a dictionary word at index $0 \leq j < 4096$ of length $w = 2w' + 4, 0 \leq w' \leq 7$, store 16 bits: `1` $\|$ [12-bit encoding of $j$] $\|$ [3-bit encoding of $w'$]
- To encode 2 lower-ASCII characters $z_1 z_2$, store 16 bits: `00` $\|$ [7-bit encoding of $z_1$] $\|$ [7-bit encoding of $z_2$]
- To encode 1 byte $z$, store 16 bits: `01000000` $\|$ [8-bit encoding of $z$]

### 4.4 Discussion

The main drawback of the fixed dictionary mitigation technique is that in practice it achieves relatively poor—albeit non-zero—compression compared with adaptive compression techniques. However, it does not rely on application-dependent or heuristic techniques for separating secrets.

#### Acknowledgements

The authors acknowledge support by Australian Research Council (ARC) Discovery Project DP130104304.

## References

1. Fielding, R., Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230 (Proposed Standard) (2014)
2. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard) (2008) Updated by RFCs 5746, 5878, 6176.
3. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Koblitz, N., ed.: CRYPTO'96. Volume 1109 of LNCS., Springer (1996) 104–113
4. Kelsey, J., Schneier, B., Wagner, D., Hall, C.: Side channel cryptanalysis of product ciphers. In Quisquater, J.J., Deswarte, Y., Meadows, C., Gollmann, D., eds.: ESORICS'98. Volume 1485 of LNCS., Springer (1998) 97–110

5. Hutter, M., Mangard, S., Feldhofer, M.: Power and EM attacks on passive 13.56 mhz RFID devices. In Paillier, P., Verbauwhede, I., eds.: CHES 2007. Volume 4727 of LNCS., Springer (2007) 320–333
6. Kelsey, J.: Compression and information leakage of plaintext. In Daemen, J., Rijmen, V., eds.: FSE 2002. Volume 2365 of LNCS., Springer (2002) 263–276
7. Rizzo, J., Duong, T.: The CRIME attack (2012) Presented at ekoparty '12. http://goo.gl/mlw1X1.
8. The Chromium Projects: (SPDY) http://dev.chromium.org/spdy.
9. Trustworthy Internet Movement: SSL Pulse (2014) https://www.trustworthyinternet.org/ssl-pulse/.
10. Gluck, Y., Harris, N., Prado, A.: SSL, gone in 30 seconds: A BREACH beyond CRIME. In: Black Hat USA 2013. (2013)
11. Belshe, M., Peon, R., Thomson, M.: Hypertext Transfer Protocol version 2 (2014) Internet-Draft. http://tools.ietf.org/html/draft-ietf-httpbis-http2-16.
12. Kelley, J., Tamassia, R.: Secure compression: Theory & practice. Cryptology ePrint Archive, Report 2014/113 (2014) http://eprint.iacr.org/2014/113.
13. Akavia, A., Goldwasser, S., Vaikuntanathan, V.: Simultaneous hardcore bits and cryptography against memory attacks. In Reingold, O., ed.: TCC 2009. Volume 5444 of LNCS., Springer (2009) 474–495
14. Alwen, J., Dodis, Y., Wichs, D.: Leakage-resilient public-key cryptography in the bounded-retrieval model. In Halevi, S., ed.: CRYPTO 2009. Volume 5677 of LNCS., Springer (2009) 36–54
15. Dziembowski, S., Pietrzak, K.: Leakage-resilient cryptography. In: 49th FOCS, IEEE Computer Society Press (2008) 293–302
16. Naor, M., Segev, G.: Public-key cryptosystems resilient to key leakage. In Halevi, S., ed.: CRYPTO 2009. Volume 5677 of LNCS., Springer (2009) 18–35
17. Alawatugoda, J., Stebila, D., Boyd, C.: Protecting encrypted cookies from compression side-channel attacks (full version) (2014) Cryptology ePrint Archive, Report 2014/724. http://eprint.iacr.org/2014/724.
18. Pike, J.: Text compression using a 4 bit coding scheme. The Computer Journal **24** (1980) 324–330
19. Sanfilippo, S.: Smaz: Small strings compression library (2009) https://github.com/antirez/smaz.

## A Relations and separations between security notions

This section briefly gives the intuition for the proofs of the relations and separations of the security notions; details appear in the full version [17].

- IND-CPA $\implies$ CCI: A (non-compressing) IND-CPA-secure symmetric-key encryption scheme provides indistinguishability of any pair of equal-length chosen messages, including messages involving a cookie. The proof proceeds by a hybrid argument, making the cookie used in each query made by the adversary to its $E_1$ oracle independent of the secret bit $b$.
- CCI $\not\implies$ IND-CPA: A degenerate scheme that uses a separating-secrets filter to extract secret cookies then encrypt the cookies but not the non-cookie data is CCI-secure but not IND-CPA-secure for the whole message.

– CCI $\implies$ RCI: A straightforward simulation: an adversary who cannot distinguish between encryptions of equal-length cookies of its choosing can also not distinguish between encryptions of randomly chosen equal-length cookies.
– RCI $\not\implies$ CCI: A counterexample is constructed that uses a separating-secrets filter: an extra ciphertext component $c_2$ is added, consisting of a point function applied to the separated secrets, where the point function is 1 on a single, publicly known cookie value $z$. With high probability, two randomly chosen cookies will not match $z$, so $c_2$ carries no useful information and the scheme is RCI-secure, but a CCI adversary can choose one cookie that matches $z$ and one that does not, so $c_2$ allows distinguishing of the chosen cookies.
– RCI $\implies$ CR: A straightforward simulation: an adversary who recovers a cookie given only ciphertexts easily distinguishes encryptions of cookies.
– CR $\not\implies$ RCI: A counterexample is constructed: an extra ciphertext component $c_2$ is added, consisting of a random oracle applied to the message. The adversary gets encryptions of $m'\|ck\|m''$ for $m', m''$ of its choice; without querying the random oracle on exactly $m'\|ck\|m''$, $c_2$ provides no information to the adversary, so the scheme is CR-secure. However, an RCI adversary can check the random oracle on the two given random cookies, so $c_2$ allows distinguishing of the given random cookies.

## B Proof of CCI security of separating-secrets technique

*Proof of Theorem 1.* The proof proceeds in a sequence of games, using a hybrid approach. Each Game $i$ proceeds as in the original CCI security experiment, except that the queries to $E_1$ are answered as in Figure 7. Let $\mathrm{Adv}^i$ denote the probability that game $i$ outputs 1.

$\underline{E_1(m', m'')}$
1: **if** query $\# \leq i$ **then**
2: **return** $\Pi.\mathrm{Enc}_k(\mathrm{SS}_{f,\Gamma}(m'\|ck_0\|m''))$
3: **else if** query $\# > i$ **then**
4: **return** $\Pi.\mathrm{Enc}_k(\mathrm{SS}_{f,\Gamma}(m'\|ck_b\|m''))$

**Fig. 7.** Oracle $E_1$ used in Game $i$ in proof of Theorem 1.

*Game 0.* This is the original CCI security game for $\Pi$. By definition, $\mathrm{Adv}^{\mathsf{CCI}}_{\Pi\circ\mathrm{SS}_{f,\Gamma},\mathcal{CK}}(\mathcal{A}) = \mathrm{Adv}^0$ .

*Transition from Game $(i-1)$ to Game $i$, $1 \leq i \leq q$.* Each hybrid transition changes how one query is answered; if the adversary's behaviour differs because of the change in answering the query, we can construct a simulator $\mathcal{B}_i$ that wins the IND-CPA game for $\Psi$, as shown in Figure 8. When the IND-CPA challenger uses

$\underline{\mathcal{B}_i^{\mathcal{A},E}()}$

1: $(ck_0, ck_1, st) \xleftarrow{\$} \mathcal{A}^{E_2}()$ s.t. $|ck_0| = |ck_1|$
2: $\hat{b} \xleftarrow{\$} \{0,1\}$
3: $b' \xleftarrow{\$} \mathcal{A}^{E_1,E_2}(ck_0, ck_1, st)$
4: **return** $b'$

$\underline{E_2(m)}$

1: **return** $E(SS_{f,\Gamma}.\text{Comp}(m))$

$\underline{E_1(m', m'')}$

1: **if** query # $< i$ **then**
2: **return** $E(SS_{f,\Gamma}.\text{Comp}(m'\|ck_0\|m''))$
3: **else if** query # $= i$ **then**
4: $pt\|\widetilde{pt_{ns}} \leftarrow SS_{f,\Gamma}.\text{Comp}(m'\|ck_{\hat{b}}\|m'')$
5: $pt'\|\widetilde{pt_{ns}}' \leftarrow SS_{f,\Gamma}.\text{Comp}(m'\|ck_0\|m'')$
6: Give $(pt\|\widetilde{pt_{ns}}, pt'\|\widetilde{pt_{ns}}')$ to IND-CPA challenger
7: Receive $c^*$ from IND-CPA challenger
8: **return** $c^*$
9: **else if** query # $> i$ **then**
10: **return** $E(SS_{f,\Gamma}.\text{Comp}(m'\|ck_{\hat{b}}\|m''))$

**Fig. 8.** Simulator $\mathcal{B}_i$ used in the proof of Theorem 1

$b = 0$, $c^*$ is the encryption of the separating-secrets compression of $m'\|ck_{\hat{b}}\|m''$, so $\mathcal{B}_i$ is playing game $(i-1)$ with $\mathcal{A}$. When the IND-CPA challenger uses $b = 1$, $c^*$ is the encryption of the separating-secrets compression of $m'\|ck_0\|m''$, so $\mathcal{B}_i$ is playing game $i$ with $\mathcal{A}$. Since $f$ is safe for $\mathcal{CK}$, the separating-secrets compressions of $m'\|ck_0\|m''$ and $m'\|ck_1\|m''$ have the same length, and thus the pair of chosen messages given from the simulator in $E_1$ to the IND-CPA challenger is valid according to the IND-CPA experiment. Thus, $\left|\text{Adv}^{i-1} - \text{Adv}^{i}\right| \leq \text{Adv}^{\mathsf{IND\text{-}CPA}}_{\Psi}(\mathcal{B}_i^{\mathcal{A}})$ .

*Analysis of Game q.* Since the adversary's view is independent of $b$ in Game $q$, we have $\text{Adv}^{q} = 0$ .

*Conclusion.* Combining the above results, we have $\text{Adv}^{\mathsf{CCI}}_{\Psi,\mathcal{CK}}(\mathcal{A}) \leq \sum_{i=1}^{q} \text{Adv}^{\mathsf{IND\text{-}CPA}}_{\Psi}(\mathcal{B}_i^{\mathcal{A}}) = q \cdot \text{Adv}^{\mathsf{IND\text{-}CPA}}_{\Psi}(\mathcal{B}^{\mathcal{A}})$ (with a small abuse of notation in creating a single $\mathcal{B}$ from the disparate $\mathcal{B}_i$). □

# C Analysis of security of fixed-dictionary technique

## C.1 Probability bounds, no prefix/suffix

In this section, we compute the amount of information given to the adversary from knowing the length of the compressed cookie, without any adversarially chosen prefix or suffix. This can be computed by calculating the amount of information given by knowing how many substrings of the cookie appear in the dictionary. For the analysis, we treat $\mathcal{D}$ as a set of strings. Proofs for results in this section appear in the full version of the paper [17].

First we calculate the probability that a given string is a substring of a randomly chosen cookie.

**Lemma 1.** *Let $x \in \Omega^w$ be a word, and let $ck \xleftarrow{\$} \Omega^n = \mathcal{CK}$ be a random string of $n$ characters. Then* $\Pr(x \preceq ck) \leq 1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}$ .

We now compute that probability that one of a set of given strings is a substring of a randomly chosen cookie:

**Lemma 2.** *Let $\mathcal{D} \subseteq \Omega^w$ with $|\mathcal{D}| = d$ be a dictionary of $d$ words of $w$ characters. Let $ck \xleftarrow{\$} \Omega^n = \mathcal{CK}$ be a random string of $n$ characters. Then*

$$\Pr(\exists x \in \mathcal{D} : x \preceq ck) \leq d\left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}\right) \ .$$

Recall the definition of conditional entropy for random variables $X$ and $Y$:

$$\begin{aligned} H(Y \mid X) &= \sum_{x \in \text{supp}(X)} \Pr(X = x) H(Y \mid X = x) \\ &= -\sum_{x \in \text{supp}(X)} \Pr(X = x) \\ &\quad \cdot \sum_{y \in \text{supp}(Y)} \Pr(Y = y \mid X = x) \log_2 \Pr(Y = y \mid X = x) \ . \end{aligned}$$

We now compute the amount of entropy about the cookie given knowledge about the number of substrings of the cookie that appear in the dictionary:

**Lemma 3.** *Fix $\mathcal{D}$. Let $\#\text{SUB}(ck)$ denote the number of substrings of $ck$ that appear in $\mathcal{D}$. Suppose $CK$ is a uniform random variable on $\mathcal{CK}$. Then*

$$\begin{aligned} H(CK \mid \#\text{SUB}(CK)) \geq &\left(1 - d\left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}\right)\right) \\ &\cdot \log_2 \left(|\mathcal{CK}| - |\mathcal{CK}| \cdot d\left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}\right)\right) \ . \end{aligned}$$

For example, if we have 16-byte cookies ($\mathcal{CK} = \{\texttt{0x00}, \ldots, \texttt{0xFF}\}^{16}$), and the dictionary $\mathcal{D}$ is a set of $d = 4096$ words of length $w = 4$ bytes, then

$$H(CK \mid \#\text{SUB}(CK)) \geq 127.998395 \ .$$

Concluding our analysis of the information learned given to the adversary without any adversarially chosen prefix or suffix, we give a bound on the amount of entropy about the cookie given the length of the compressed cookie:

**Lemma 4.** *Fix $\mathcal{D}$ with $d$ words of length $w$ over character set $\Omega$. Denote the length of a cookie $ck$ compressed with dictionary $\mathcal{D}$ by $\text{COMPLEN}(ck) = |\text{FD}_{\mathcal{D},w,\ell}.\text{Comp}(ck)|$. Suppose $CK$ is a uniform random variable on $\mathcal{CK}$. Then*

$$\begin{aligned} H(CK \mid \text{COMPLEN}(CK)) &\geq H(CK \mid \#\text{SUB}(CK)) \\ &\geq \left(1 - d\left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}\right)\right) \\ &\quad \cdot \log_2 \left(|\mathcal{CK}| - |\mathcal{CK}| \cdot d\left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}\right)\right) \ . \end{aligned}$$

Lemma 4 follows from the data processing inequality and Lemma 3.

### C.2 Probability bounds, prefix/suffix

Suppose $CK$ is a uniform random variable on $\mathcal{CK} = \Omega^n$. We know that $H(CK) = n\log_2(|\Omega|)$. Trivially, $H(CK \mid CK_1) = (n-1)\log_2(|\Omega|)$, where $CK_1$ is the first character of $CK$. Similarly, $H(CK \mid CK_{1:a}) = (n-a)\log_2(|\Omega|)$ and finally $H(CK \mid CK_{1:a}, CK_{n-b:b}) = (n-a-b)\log_2(|\Omega|)$.

Consider the following CRIME-like attack on the beginning of the cookie. Let $\mathcal{D}$ be a dictionary with $d$ words of length $w$ over character set $\Omega$. Let $ck \in \Omega^n$. Let $O(\cdot)$ be an oracle that, upon input $a$ of length $w-m$, with $1 \le m \le w-1$, returns 1 if and only if $a \| ck_{1:m} \in \mathcal{D}$.

The CRIME-like attack works as follows:

1. For each $x \in \mathcal{D}$, query $x_{1:w-1}$ to the oracle. If a query for $x_{1:w-1}$ returns 1, then it is known that $ck_{1:1} \in Z_1 = \{z : x_{1:w-1} \| z \in \mathcal{D}\}$. If no query returns 1, then return $\emptyset$.
2. For $m = 2, \ldots, w-1$: For each $x \in \mathcal{D}$ such that $x_{w-m} \in Z_{m-1}$, query $x_{1:w-m}$ to the oracle. If a query for $x_{1:w-m}$ returns 1, then it is known that $ck_{1:m} \in Z_m = \{z_1 z_2 \ldots z_m : x_{1:w-m} \| z_1 z_2 \ldots z_m \in \mathcal{D}\}$. If no query returns 1, then return $Z_1, \ldots, Z_{m-1}$.
3. Return $Z_1, \ldots, Z_{w-1}$.

A corresponding attack on the suffix is obvious.

Let $\mathrm{CRIMEpre}(ck)$ denote the output obtained from running the above prefix CRIME attacks on $ck$, $\mathrm{CRIMEsuf}(ck)$ denote the output from the corresponding suffix attack. Let $\mathrm{CRIME}(ck) = (\mathrm{CRIMEpre}(ck), \mathrm{CRIMEsuf}(ck))$.

Noting that in the best case the CRIME attack allows the attacker to learn the first $w-1$ and the last $w-1$ characters of the cookie, some trivial lower bounds are:

$$H(CK_{1:w-1} \mid \mathrm{CRIME}(CK)) \ge 0$$
$$H(CK_{n-w+1:w-1} \mid \mathrm{CRIME}(CK)) \ge 0$$

However, the CRIME attack provides no information about the remaining characters, so $I(CK_{1:w-1}, CK_{w:n-w+1}) = 0$ and $I(CK_{1:n-w+1}, CK_{n-w+1:w-1}) = 0$, and thus $H(CK_{w:n-w+2} \mid \mathrm{CRIME}(CK), \mathrm{COMPLEN}(CK)) = H(CK_{w:n-w+2} \mid \mathrm{COMPLEN}(CK))$.

Finally, we have that

$$\begin{aligned} &H(CK \mid \mathrm{CRIME}(CK), \mathrm{COMPLEN}(CK)) \\ &\quad \ge H(CK_{1:w-1} \mid \mathrm{CRIMEpre}(CK)) + H(CK_{w:n-w+2} \mid \mathrm{COMPLEN}(CK)) \\ &\qquad + H(CK_{n-w+1:w-1} \mid \mathrm{CRIMEsuf}(CK)) \\ &\quad \ge 0 + H(CK_{w:n-w+2} \mid \mathrm{COMPLEN}(CK)) + 0 \end{aligned}$$

and we can obtain a lower bound on $H(CK_{w:n-w} \mid \mathrm{COMPLEN}(CK))$ using Lemma 4.