Securing Nonintrusive Web Encryption through Information Flow

Lantian Zheng
Google Inc.
zlt@google.com

Andrew C. Myers
Computer Science Department
Cornell University
andru@cs.cornell.edu

Abstract

This paper proposes a nonintrusive encryption mechanism for protecting data confidentiality on the Web. The core idea is to encrypt confidential data before sending it to untrusted sites and use keystores on the Web to manage encryption keys without intervention from users. A formal language-based information flow model is used to prove the soundness of the mechanism.

Categories and Subject Descriptors D.4.6 [*Security and Protection*]: Cryptographic controls; Information flow controls

General Terms Languages, Security

1. Introduction

People store increasing amounts of personal data (emails, contacts, calendars, documents, photos and more) on the Web. Protecting the confidentiality of online personal data is critical. It is also challenging because many users have a high tolerance for insecurity, but a low tolerance for inconvenience. Websites share user-generated data with business partners and have vulnerabilities that may lead to information leaks, yet users ignore these risks and send confidential data to untrusted sites in order to use their services.

Our goal is to design a protection mechanism that is *nonintrusive*, in the sense that it does not blindly prevent users from accessing web services that on the surface involve sending confidential data to untrusted sites, and it requires little user intervention. The solution exploits a simple observation: many websites only need to store and/or forward users' data without interpreting or processing the data. For example, an online album service only needs to store photos on the server side. Therefore, if the album site stores a photo simply as a byte array, it is possible for users to store encrypted photos on the album site without affecting usability of the service. When accessing the album site, the user's browser can retrieve encrypted photos from the site, and decrypt and display the photos to the user. The challenge is to handle encryption and decryption with little user intervention.

To address this challenge, we propose a symmetric encryption scheme with transparent key generation and management. Keys are stored on the Web so that they are world-accessible. Then, encrypted data is augmented with the location of the key so that the receiver of encrypted data knows where to get the key. As a result, end users are spared of the burden of generating, storing and securing encryption keys. Web applications can encrypt and decrypt data transparently, without affecting usability.

This *nonintrusive encryption* technique alone cannot ensure data confidentiality. We still need to ensure that encryption keys are not exposed to untrusted sites, that confidential data is not sent to untrusted sites in cleartext, and that cryptographic primitives do not introduce implicit flows [9]. These requirements can be satisfied using static information flow control [9, 16, 21], which labels data with security levels and uses static program analysis to ensure the absence of insecure information flows: high-confidentiality data affecting low-confidentiality data. We imagine that the technique can be deployed on both the user's browser and on websites to check web application code at load time.

This paper combines the nonintrusive encryption technique and static information flow control, and presents a sequential security-typed language (called Sweb) with cryptographic primitives. The type system of Sweb ensures that well-typed code does not explicitly or implicitly assign cleartext confidential data to untrusted storage locations (sites), satisfying a strong notion of confidentiality—noninterference [11], albeit under some assumptions about the strength of the encryption algorithm.

Previous work [4, 24] has shown that a security-typed language with encryption primitives can enforce noninterference. These type systems have treated the result of encryption as public, which only makes sense if the encryption key is as confidential as the plaintext. This constraint may be too strong for the web environment where keys are stored online. In reality, a ciphertext is not necessarily made public. As a result, it is possible to relax the constraint. In the album site example, suppose Alice's browser connects to the album site through SSL. Then the encrypted photo is only readable by Alice and the album site. As a result, Alice's browser can store the encryption key on some keystore even if Alice does not trust the keystore site to access her photos, but does trust that the keystore site and the album site will not collude to leak her photos. The type system of Sweb formalizes this insight and results in more permissive typing than previous work.

The idea of splitting a secret into multiple shares for high confidentiality is well known [23, 22]. Our contribution is to apply the idea to typing the encryption primitive, formalize the confidentiality guarantee and prove correctness by showing that the type system enforces noninterference.

The rest of this paper is organized as follows. Section 2 describes the nonintrusive encryption technique. Section 3 introduces the Sweb language. Section 4 discusses information flow control enhanced with encryption. Section 5 describes the type system of Sweb, and shows that it can enforce noninterference. Section 6 covers related work, and section 7 concludes.

PLAS'08, June 8, 2008, Tucson, Arizona, USA.

2. Nonintrusive encryption

We propose the following nonintrusive encryption technique.

- Some websites, presumably more trusted than others, provide *keystore* services. A keystore maps identifiers to symmetric encryption keys, and a keystore service $\mathcal{K}$ provides two APIs: *newkey*($\mathcal{K}$) returns a pair $\mathtt{i} : k$ where k is a fresh key, and $\mathtt{i}$ is the identifier of k; $\mathcal{K}(\mathtt{i})$ returns the key mapped to $\mathtt{i}$ in $\mathcal{K}$. Each keystore service is publicly accessible through a name K.
- The encryption primitive has the form *encrypt*(d, K), which obtains a new key k with identifier $\mathtt{i}$ from keystore K, encrypts d with k to obtain the ciphertext c and returns $c.K.\mathtt{i}$ as the encryption result.
- The decryption primitive has the form *decrypt*($c.K.\mathtt{i}$), which retrieves the key k from keystore K with identifier $\mathtt{i}$, decrypts c with k and returns the plaintext.

Interestingly, this scheme does not provide a key generation primitive and every encryption operation implicitly obtains a new key from the keystore being used. The implicit key generation makes key management transparent and less error-prone. In addition, it practically achieves the same effect as the *IND-CPA security* (indistinguishability under chosen-plaintext attack) [7], since an attacker can encrypt a chosen plaintext with the same key only once.

On the other hand, the treatment comes with the limitation that keys cannot be reused. But this limitation is bearable in the web environment, where storage is never the bottleneck. Suppose Alice makes ten encryptions every day, and each key takes 1000 bytes. Then all the keys that Alice needs in her lifetime take less than 500 megabytes of storage.

In addition, the encryption scheme is easy to deploy. With a secure email service (or other existing secure storage services), a browser extension can implement a keystore service straightforwardly. For example, suppose Alice trusts her Gmail account to keep mail confidential. Keystore `alice@gmail.com` can be implemented as follows:

- To implement *newkey*(`alice@gmail.com`), her browser generates a new key k and a random identifier $\mathtt{i}$, sends an email $\langle \mathtt{subject} : \mathtt{i}, \mathtt{body} : k \rangle$ to `alice@gmail.com` through SSL (luckily, `gmail.com` can be accessed through `https`) and returns the pair $\mathtt{i} : k$.
- To implement `alice@gmail.com`(i), her browser simply retrieves Alice's email with subject $\mathtt{i}$ from `gmail.com` and returns the email body.

Note that with session cookies or saved password, Alice's browser can access her Gmail account without her intervention, and thus the encryption and decryption operations can be totally transparent. As a result, Alice would be able to use an online album site safely, perhaps not even realizing that her photos are encrypted before being sent to the site and decrypted before being displayed in the browser.

3. The language

The Web allows users to store and retrieve data, and invoke computations, all through a global name space (URLs). These core web functionalities can be modeled by a simple imperative language (Sweb) with shared memory and functions. For example, let memory location m_b represent user's browser output and let $m_{\mathtt{foo.com}}$ represent the web page `http://foo.com`. Then the assignment statement $m_b := !m_{\mathtt{foo.com}}$ models a browser access to the URL `http://foo.com`.

Suppose function name $f_{\mathtt{foo.com/cgi}}$ represents the CGI program at `http://foo.com/cgi`. Then statement `call` $f_{\mathtt{foo.com/cgi}}$ models accessing the URL `http://foo.com/cgi`. The last statement of $f_{\mathtt{foo.com/cgi}}$ should be $m_b := e$, returning the result page to the user's browser. Furthermore, the following program models invoking the CGI program with two arguments (accessing the URL `http://foo.com/cgi?a1=v1&a2=v2`):

$$m_{\mathtt{foo.a1}} := \mathtt{v1};$$
$$m_{\mathtt{foo.a2}} := \mathtt{v2};$$
$$\mathtt{call}\ f_{\mathtt{foo.com/cgi}}$$

while the code of $f_{\mathtt{foo.com/cgi}}$ retrieves the arguments from memory locations $m_{\mathtt{foo.a1}}$ and $m_{\mathtt{foo.a2}}$. Note that Sweb is sequential and does not model concurrent accesses to a URL. This treatment of function arguments and results is crude but adequate for our purposes.

Invoking a remote function through a URL can also be used for communication between websites, and thus Sweb can express web applications involving multiple websites, as shown in Section 3.3.

In Sweb, a simple dereferencing expression $!m$ might represent reading information from a remote website. Static information flow analysis can prevent a good machine from running code that leaks confidential information. But a compromised machine might still try to read confidential data from a remote host and then leak the data.

We assume that a run-time access control mechanism is deployed so that read requests from untrusted machines for confidential data would be rejected. In particular, a keystore would not send keys to untrusted machines, so a compromised machine is not able to obtain confidential data by corrupting code execution. Therefore, we can assume that code execution is safe on any server machine. Note that this assumption would not be valid if data integrity interacted with confidentiality as with robust declassification [27]. However, Sweb considers neither declassification nor data integrity.

References	r	$::=$	$m \mid f \mid K$
Values	v	$::=$	$n \mid m \mid c.K.\mathtt{i}$
Expressions	e	$::=$	$v \mid !e \mid \mathit{decrypt}(e) \mid e_1 + e_2$
Statements	s	$::=$	$e_1 := e_2 \mid e_1 := \mathit{encrypt}(e_2, K)$
		$\mid$	$\mathtt{if}\ e\ \mathtt{then}\ s_1\ \mathtt{else}\ s_2$
		$\mid$	$\mathtt{skip} \mid s_1; s_2 \mid \mathtt{call}\ f$

Figure 1. Syntax of the Sweb language

3.1 Syntax

The syntax of the Sweb language is shown in Figure 1. A reference r may be a memory location m, a function name f, or a keystore name K. In Sweb, a value may be an integer n, a memory location m or an encrypted value $c.K.\mathtt{i}$ where c is a ciphertext, and $\mathtt{i}$ is a key identifier. An expression may be a value v, a dereference expression $!e$ (only dereferencing memory locations), or a decrypt expression *decrypt*(e). For a technical reason (avoiding expressions with side effects), the encryption primitive is formalized as a statement $e_1 :=$ *encrypt*(e_2, K), which encrypts the value of e_2 using a key in keystore K and then assigns the encrypted value to the memory location that is the result of e_1.

Other statements of Sweb include the assignment $e_1 := e_2$, the conditional statement $\mathtt{if}\ e\ \mathtt{then}\ s_1\ \mathtt{else}\ s_2$, the sequence $s_1; s_2$, the skip statement `skip`, and the call statement $\mathtt{call}\ f$. The call statement supports recursive function calls, and thus Sweb does not include a loop statement.

3.2 Operational semantics

Let W represent a state of the Web, which is a finite map, mapping memory locations to values, function names to programs, and keystore names to keystores. A Sweb program s is evaluated in a

$$\text{(E1)}\quad \frac{W(m) = v}{\langle !m,\, W\rangle \Downarrow v}$$

$$\text{(E2)}\quad \langle v,\, W\rangle \Downarrow v$$

$$\text{(E3)}\quad \frac{\langle e,\, W\rangle \Downarrow c.K.\mathtt{i} \quad W(K)(\mathtt{i}) = k \quad \mathcal{D}(c,\, k) = v}{\langle \mathit{decrypt}(e),\, W\rangle \Downarrow v}$$

$$\text{(E4)}\quad \frac{\langle e_1,\, W\rangle \Downarrow n_1 \quad \langle e_2,\, W\rangle \Downarrow n_2}{\langle e_1 + e_2,\, W\rangle \Downarrow n_1 + n_2}$$

$$\text{(S1)}\quad \frac{\langle e,\, W\rangle \Downarrow n \quad n > 0}{\langle \mathtt{if}\ e\ \mathtt{then}\ s_1\ \mathtt{else}\ s_2,\, W\rangle \longmapsto \langle s_1,\, W\rangle}$$

$$\text{(S2)}\quad \frac{\langle e,\, W\rangle \Downarrow n \quad n \leq 0}{\langle \mathtt{if}\ e\ \mathtt{then}\ s_1\ \mathtt{else}\ s_2,\, W\rangle \longmapsto \langle s_2,\, W\rangle}$$

$$\text{(S3)}\quad \frac{W(f) = s}{\langle \mathtt{call}\ f,\, W\rangle \longmapsto \langle s,\, W\rangle}$$

$$\text{(S4)}\quad \frac{\langle s_1,\, W\rangle \longmapsto \langle s_1',\, W'\rangle}{\langle s_1; s_2,\, W\rangle \longmapsto \langle s_1'; s_2,\, W'\rangle}$$

$$\text{(S5)}\quad \langle \mathtt{skip}; s,\, W\rangle \longmapsto \langle s,\, W\rangle$$

$$\text{(S6)}\quad \frac{\langle e_1,\, W\rangle \Downarrow m \quad \langle e_2,\, W\rangle \Downarrow v \quad W(K) = \mathcal{K} \quad \mathit{newkey}(\mathcal{K}) = \mathtt{i}{:}k \quad \mathcal{E}(v,\, k) = c \quad W' = W[K \mapsto \mathcal{K}[\mathtt{i} \mapsto k]][m \mapsto c.K.\mathtt{i}]}{\langle e_1 := \mathit{encrypt}(e_2,\, K),\, W\rangle \longmapsto \langle \mathtt{skip},\, W'\rangle}$$

$$\text{(S7)}\quad \frac{\langle e_1,\, W\rangle \Downarrow m \quad \langle e_2,\, m\rangle \Downarrow v}{\langle e_1 := e_2,\, W\rangle \Downarrow \langle \mathtt{skip},\, W[m \mapsto v]\rangle}$$

Figure 2. Operational semantics of Sweb

web state, resulting in new web states. Thus, a small evaluation step is a transition from configuration $\langle s,\, W\rangle$ to another configuration $\langle s',\, W'\rangle$. Because Sweb expressions have no side effects, we use the notation $\langle e,\, W\rangle \Downarrow v$ to mean that evaluating e in web state W results in the value v. The operational semantics of Sweb is shown in Figure 2.

The notation $W(r)$ represents the entity mapped to r. The notation $W[r \mapsto v]$ (or $W[r \mapsto \mathcal{K}]$) denotes the web state obtained by assigning value v (or keystore $\mathcal{K}$) to r in W.

Most evaluation rules are standard. Rule (E3) evaluates decryption expressions. The key k for decrypting $c.K.\mathtt{i}$ is retrieved from $W(K)$ using identifier $\mathtt{i}$. Applying the decryption function $\mathcal{D}$ to the ciphertext c and key k results in v.

Rule (S6) is used to evaluate encryption statement $e_1 := \mathit{encrypt}(e_2,\, K)$. Suppose the result of e_1 is memory location m, and the result of e_2 is v, and $W(K)$ is the keystore $\mathcal{K}$, which is a tuple $\langle \overline{\mathtt{i}{:}k},\, \mathtt{T}\rangle$, where $\overline{\mathtt{i}{:}k}$ is a list of new identifier-key pairs that have not been used for encryption, and $\mathtt{T}$ is a key table mapping identifiers to keys that have been used to encrypt some value. The auxiliary function $\mathit{newkey}(\mathcal{K})$ returns the first identifier-key pair in $\overline{\mathtt{i}{:}k}$, and $\mathcal{K}[i \mapsto k]$ returns the keystore obtained by removing $i : k$ from the new key list and inserting it into the used key table of $\mathcal{K}$. This keystore formalization avoids introducing a random key generator that would complicate the proof of noninterference.

Suppose $\mathit{newkey}(\mathcal{K}) = \mathtt{i} : k$. Then $\mathcal{E}(v,\, K)$ encrypts v with key k and results in a ciphertext c. We assume the encryption algorithm $\mathcal{E}$ is strong enough such that no information about v or k can be inferred from the ciphertext c. Again, to simplify the noninterference proof, we assume that $\mathcal{E}$ is deterministic. This assumption does not make the system subject to chosen-plaintext attacks because each key can be used only once for encryption.

In rule (S6), the new web state W' is obtained by assigning the encrypted value $c.K.\mathtt{i}$ to m, and the keystore $\mathcal{K}[\mathtt{i} \mapsto k]$ to K.

3.3 Example

As simple as it is, Sweb is expressive enough to model some real-world applications. Suppose Alice wants to buy something from an on-line store `foo.com`. To place the order, she needs to send her address and her credit card number to `foo.com`, which then contacts `visa.com` to charge her card and `ups.com` to ship the order. Suppose Alice does not trust `foo.com` to protect the confidentiality of her address and card number. Assuming `ups.com` and `visa.com` provide keystore services, the transaction can still be performed in the following way:

- After Alice fills in the order form, her browser gets a new key k_1 with identifier $\mathtt{i}_1$ from `ks.visa.com` (the keystore of `visa.com`), encrypts her card number (modeled by a memory location in Sweb) with k_1, and then sends $c_{\mathtt{card}}.K_{\mathtt{ks.visa.com}}.\mathtt{i}_1$ to `foo.com`. Similarly, Alice's address is encrypted with a key k_2 from `ks.ups.com`, and $c_{\mathtt{addr}}.K_{\mathtt{ks.ups.com}}.\mathtt{i}_2$ is sent to `foo.com`. Then $f_{\mathtt{foo.com/order}}$ is called to handle the order. The following code models the process:

  $m_{\mathtt{foo.com/order?a1}} := \mathit{encrypt}(!m_{\mathtt{cc}},\, K_{\mathtt{ks.visa.com}})$
  $m_{\mathtt{foo.com/order?a2}} := \mathit{encrypt}(!m_{\mathtt{addr}},\, K_{\mathtt{ks.ups.com}})$
  $\mathtt{call}\ f_{\mathtt{foo.com/order}}$

- The code of $f_{\mathtt{foo.com/order}}$ processes an order and is shown as follows:

  $m_{\mathtt{visa.com/charge?account}} := !m_{\mathtt{foo.com/order?a1}};$
  $m_{\mathtt{visa.com/charge?amount}} := !m_{\mathtt{amount}}$
  $\mathtt{call}\ f_{\mathtt{visa.com/charge}};$
  $m_{\mathtt{ups.com/ship?addr}} := !m_{\mathtt{foo.com/order?a2}};$
  $\mathtt{call}\ f_{\mathtt{ups.com/ship}};$
  $m_b := !m_{\mathtt{track\text{-}num}}$

  The code first sends the encrypted card number and the charge amount to `visa.com` and invokes the charge function. Then the encrypted address is sent to `ups.com` (perhaps by printing $c_{\mathtt{addr}}.K_{\mathtt{ks.ups.com}}.\mathtt{i}_2$ on a UPS shipping label). The UPS shipping function returns a tracking number ($m_{\mathtt{track\text{-}num}}$), which is returned to Alice's browser (m_b).

  Interestingly, the code of $f_{\mathtt{foo.com/order}}$ can remain the same no matter whether the values stored at $m_{\mathtt{foo.com/order?a1}}$ and $m_{\mathtt{foo.com/order?a2}}$ are encrypted. This is generally the case because an untrusted site only needs to store and/or forward encrypted values. This property could allow an untrusted site to work regardless of whether encryption is being used.

- The code of $f_{\mathtt{visa.com/charge}}$ is as follows:

  $m_{\mathtt{card}} := \mathit{decrypt}(m_{\mathtt{visa.com/charge?account}});$
  $!m_{\mathtt{card}} := !!m_{\mathtt{card}} + !m_{\mathtt{visa.com/charge?amount}}$

  This code first decrypts the encrypted memory location representing Alice's card number and assigns the memory location to $m_{\mathtt{card}}$. Then it increments the value of $!m_{\mathtt{card}}$ by the order amount.

  Note that $f_{\mathtt{visa.com/charge}}$ runs on the server of `visa.com`, which is trusted by `ks.visa.com`, and thus can read keys from the keystore. It is important that key-retrieving requests from un-

trusted sites would be rejected by keystore `ks.visa.com`. As discussed later in Section 5, the ability of a keystore to keep keys confidential is specified by the type of the keystore and taken into account by type checking.

- The code of $f_{\texttt{ups.com/ship}}$ is as follows:

$$m_{\texttt{addr}} := \mathit{decrypt}(m_{\texttt{ups.com/ship?addr}});$$
$$\texttt{call}\ f_{\texttt{internal.ups/shipping}};$$
$$m_{\texttt{track-num}} :=! m_{\texttt{track-num}} + 1$$

  First, it decrypts Alice's address. Then it invokes an internal shipping function to process the shipping order. Finally, it increments the tracking number, simulating the creation of a new tracking number.

4. Information flow control and encryption

Information flow control prevents high-confidentiality information from flowing to low-confidentiality locations. The concepts of high and low confidentiality are determined by labeling information and memory locations with security labels from a lattice $\mathcal{L}$. Given two labels ℓ_1 and ℓ_2, if $\ell_1 \leq \ell_2$ in $\mathcal{L}$, then ℓ_1 represents a confidentiality level lower than or equal to ℓ_2. Users are labeled too. A user with label ℓ can observe any memory location with a label less than or equal to ℓ. Let L represent the confidentiality level of attackers (*low users*). Then ℓ is a low-confidentiality label if $\ell \leq L$, and a high-confidentiality label if otherwise.

For example, consider a statement $m := e$. Let ℓ_m and ℓ_e be the label of m and e, respectively. Then $\ell_e \leq \ell_m$ must hold. Otherwise, it is possible that $\ell_e \not\leq L$ and $\ell_m \leq L$, and the statement assigns a high-confidentiality value to a low-confidentiality location.

Conventional information flow analysis works reasonably well for ordinary computation, but applying it to cryptographic operations poses some challenges that have not yet been satisfactorily addressed.

4.1 Addition and encryption

Consider an addition expression $e_1 + e_2$ with label l, where ℓ_1 and ℓ_2 are the labels of e_1 and e_2, respectively. Because both the values of e_1 and e_2 affect the value of $e_1 + e_2$, we conventionally require $\ell_1 \leq \ell$ and $\ell_2 \leq \ell$ to ensure that no information about e_1 and e_2 can be leaked through their sum. Using the lattice join operation ($\sqcup$), the two constraints can be represented by $\ell_1 \sqcup \ell_2 \leq \ell$.

Encryption makes things a bit more interesting. Consider the statement $m := \mathit{encrypt}(e, K)$. Let ℓ_K be the label of keystore K, and ℓ be the label of the value of m. According to evaluation rule (S6), a new key k is used to encrypt the value of e, and k is known to only users with label as high as ℓ_K. Although the value of m is affected by k and the value of e, unlike the addition case, constraints $\ell_K \leq \ell$ and $\ell_e \leq \ell$ are not needed, because no information about the value of e and k can be inferred from the encryption result. Instead, the following constraint needed to be enforced because after encryption, the value of e can be computed from the value of m and k:

$$\ell_e \leq \ell \sqcup \ell_K$$

As discussed in Section 5, this constraint leads to more precise and permissive typing than treating the encryption result as public data.

4.2 Noninterference property

To show that information flow control is effective for protecting confidentiality, we need to define confidentiality first. A strong notion of confidentiality can be formalized in term of noninterference [11], which intuitively means that high-confidentiality inputs cannot interfere with low-confidentiality outputs.

For Sweb, the inputs of a program are just the initial web state, and any web state resulted from program execution is part of the outputs. Thus, a program s satisfies the noninterference property if evaluating s under two web states with equivalent low-confidentiality parts results in web states that also have equivalent low-confidentiality parts. In other words, low users cannot distinguish the two executions.

Clearly, the key to defining the noninterference property is to define the notion that two web states W_1 and W_2 are *low-equivalent* (written $W_1 \approx_L W_2$, meaning W_1 and W_2 have equivalent low-confidentiality parts). Without encryption, the definition is straightforward: $W_1 \approx_L W_2$ if for any reference r, $\mathit{label}(r) \leq L$ implies $W_1(r) = W_2(r)$. Notation $\mathit{label}(r)$ denotes the label of r. Specifically, $\mathit{label}(m)$ is the label of the value stored in m; $\mathit{label}(K)$ is the label of keys in K; $\mathit{label}(f)$ is a lower bound of the labels of side effects of the code of f.

With encryption, we have to consider more scenarios. Suppose $W_1(m) = c_1.K.\texttt{i}_1$ and $W_2(m) = c_2.K.\texttt{i}_2$. Suppose $c_1 \neq c_2$. There are still two cases that a low user cannot distinguish the two encrypted values. First, the low user cannot observe keystore K, and thus does not know the encryption key. Then ciphertexts c_1 and c_2 are just random bits to the low user and could appear in either execution. Second, the low user can observe keystore K, but the decryption results are low-equivalent. Thus, we have the following rules that recursively define the low-equivalent relation between values:

$$v \approx_L v \qquad \frac{\mathit{label}(K) \not\leq L}{c_1.K.\texttt{i}_1 \approx_L c_2.K.\texttt{i}_2}$$

$$\frac{\mathit{label}(K) \leq L \quad \langle \mathit{decrypt}(c_i.K.\texttt{i}),\ W\rangle \Downarrow v_i,\ i \in \{1,2\} \quad v_1 \approx_L v_2}{c_1.K.\texttt{i} \approx_L c_2.K.\texttt{i}}$$

More subtly, it is not sufficient to consider the low equivalence for each individual memory location. Consider two low locations m_1 and m_2. Suppose

$$W_1(m_1) = c.K.\texttt{i} \quad W_1(m_2) = c.K.\texttt{i}$$
$$W_2(m_1) = c.K.\texttt{i} \quad W_2(m_2) = c'.K.\texttt{i}'$$

and $c \neq c'$, and $\mathit{label}(K) \not\leq L$. Then $W_1(m_1) \approx_L W_2(m_1)$ and $W_1(m_2) \approx_L W_2(m_2)$. However, W_1 and W_2 are distinguishable to low users, because the values of m_1 and m_2 are created by the same encryption operation according to W_1, and by different encryption operations according to W_2. Furthermore, we need to consider the case that $W_1(m_1)$ and $W_1(m_2)$ are different, but they can be decrypted by low-confidentiality keys, and their decryption results are the same.

Let $W^{i,L}(m)$ denote the value obtained by decrypting $W(m)$ for i times, and each time the decryption key is low-confidentiality. Then we have the following definition:

Definition 4.1 ($W_1 \approx_L W_2$)**.** $W_1 \approx_L W_2$ if the following conditions hold:

- For any m, if $\mathit{label}(m) \leq L$, then $W_1(m) \approx_L W_2(m)$.
- For any m_1 and m_2, if $\mathit{label}(m_1) \sqcup \mathit{label}(m_2) \leq L$, then for any i, j, $W_1^{i,L}(m_1) = W_1^{j,L}(m_2)$ iff $W_2^{i,L}(m_1) = W_2^{j,L}(m_2)$.
- For any K, if $\mathit{label}(K) \leq L$, then $W_1(K) = W_2(K)$.
- For any f, $W_1(f) = W_2(f)$.

5. Security type system

In Sweb, information flow control is achieved through type checking. The type system of Sweb ensures that any well-typed program satisfies the noninterference property and cannot generate illegal information flows at run time.

$$\text{(INT)} \quad \vdash n : \texttt{int}_\ell$$

$$\text{(CIPHER)} \quad \frac{\Gamma \vdash K : \texttt{keystore}_\ell\ \texttt{ref}_\bot \quad \tau \le \ell \sqcup \ell'}{\Gamma \vdash c.K.\texttt{i} : [\tau]_{\ell'}}$$

$$\text{(ADD)} \quad \frac{\Gamma \vdash e_1 : \texttt{int}_{\ell_1} \quad \Gamma \vdash e_2 : \texttt{int}_{\ell_2}}{\Gamma \vdash e_1 + e_2 : \texttt{int}_{\ell_1 \sqcup \ell_2}}$$

$$\text{(REF)} \quad \frac{\Gamma(r) = \tau}{\Gamma \vdash r : (\tau\ \texttt{ref})_\ell}$$

$$\text{(DEREF)} \quad \frac{\Gamma \vdash e : \tau\ \texttt{ref}_\ell}{\Gamma \vdash !e : \tau \sqcup \ell}$$

$$\text{(DEC)} \quad \frac{\Gamma \vdash e : [\tau]_{\ell'}}{\Gamma \vdash \mathit{decrypt}(e) : \tau \sqcup \ell'}$$

$$\text{(ENC)} \quad \frac{\begin{array}{c}\Gamma \vdash e_1 : [\tau]_{\ell'}\ \texttt{ref}_{\ell_1} \quad \Gamma \vdash e_2 : \tau \\ \Gamma \vdash K : \texttt{keystore}_\ell\ \texttt{ref}_\bot \\ \tau \le \ell \sqcup \ell' \quad \ell \le \tau \quad \ell_1 \le \ell'\end{array}}{\Gamma \vdash e_1 := \mathit{encrypt}(e_2,\ K) : \texttt{stmt}_{\ell \sqcap \ell'}}$$

$$\text{(ASSI)} \quad \frac{\Gamma \vdash e_1 : \tau\ \texttt{ref}_\ell \quad \Gamma \vdash e_2 : \tau \quad \ell \le \tau}{\Gamma \vdash e_1 := e_2 : \texttt{stmt}_{\mathit{label}(\tau)}}$$

$$\text{(SEQ)} \quad \frac{\Gamma \vdash s_1 : \tau \quad \Gamma \vdash s_2 : \tau}{\Gamma \vdash s_1; s_2 : \tau}$$

$$\text{(SKIP)} \quad \Gamma \vdash \texttt{skip} : \texttt{stmt}_\ell$$

$$\text{(IF)} \quad \frac{\Gamma \vdash e : \texttt{int}_\ell \quad \ell \le \tau \quad \Gamma \vdash s_1 : \tau \quad \Gamma \vdash s_2 : \tau}{\Gamma \vdash \texttt{if}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2 : \tau}$$

$$\text{(FUN)} \quad \frac{\Gamma \vdash f : \texttt{stmt}_\ell\ \texttt{ref}_\bot}{\Gamma \vdash \texttt{call}\ f : \texttt{stmt}_\ell}$$

$$\text{(SUB)} \quad \frac{\Gamma \vdash t : \tau \quad \tau \le \tau'}{\Gamma \vdash t : \tau'}$$

Figure 3. Type system of Sweb

This paper does not attempt to deal with termination and timing channels. Control of these channels is largely an orthogonal problem, and partially addressed in previous work [3, 20, 29].

The types of Sweb have the following syntax:

$$\begin{array}{rrcl} \text{Base types} & \beta & ::= & \texttt{int} \mid [\tau] \mid \tau\ \texttt{ref} \\ \text{Types} & \tau & ::= & \beta_\ell \mid \texttt{keystore}_\ell \mid \texttt{stmt}_\ell \end{array}$$

A type τ can be either a labeled base type β_ℓ, a keystore type $\texttt{keystore}_\ell$ or a statement type $\texttt{stmt}_\ell$. A value with type β_ℓ has label ℓ. A keystore with type $\texttt{keystore}_\ell$ is trusted to store keys with label ℓ. A statement with type $\texttt{stmt}_\ell$ has only side effects with labels higher than or equal to ℓ.

Base types include integer type $\texttt{int}$, encrypted data type $[\tau]$ and reference type $\tau\ \texttt{ref}$. Value $c.K.\texttt{i}$ has the encrypted data type $[\tau]$ if and only if it is generated by encrypting a value with type τ.

Let Γ represent a typing assignment, mapping references to types. A typing judgment of Sweb has the form $\Gamma \vdash s : \tau$ (or $\Gamma \vdash e : \tau$), meaning that statement s (or expression e) has type τ with respect to Γ.

The typing rules of Sweb are shown in Figure 3. The interesting rules are (CIPHER), (DEC) and (ENC), while other rules are standard in terms of static information flow tracking [26, 12, 28, 6, 19].

Notation $\bot$ represents the bottom label. Suppose τ is β_ℓ. Then notation $\tau \le \ell'$ represents $\ell \le \ell'$, and notation $\tau \sqcup \ell'$ represents $\beta_{\ell \sqcup \ell'}$.

Rule (CIPHER) checks encrypted values. Suppose K is the name of a keystore with label ℓ. Then $c.K.\texttt{i}$ has type $[\tau]_{\ell'}$ if $\tau \le \ell \sqcup \ell'$ holds. The label constraint ensures that a user who is authorized to read the encrypted value and the key is also authorized to read the plaintext value with type τ.

Rule (DEC) checks decryption expressions. Intuitively, if e has type $[\tau]_{\ell'}$, then the result of $\mathit{decrypt}(e)$ should have type τ. In addition, information about the result of e can be inferred from the decryption result. Thus, $\mathit{decrypt}(e)$ has type $\tau \sqcup \ell'$, ensuring its label to be as high as ℓ'.

Rule (ENC) is used to check encryption statements. Consider statement $e_1 := \mathit{encrypt}(e_2,\ K)$. The value of e_1 is a memory location for storing the encrypted value, and e_1 has type $[\tau]_{\ell'}\ \texttt{ref}_{\ell_1}$. The keystore reference K has type $\texttt{keystore}_\ell\ \texttt{ref}_\bot$. The premise $\tau \le \ell \sqcup \ell'$ is based on the same reasoning as in rule (CIPHER): putting the ciphertext and the key together can recover the original value with type τ. The premise $\ell_1 \le \ell'$ is standard, protecting information about e_1 from being leaked through the assignment to the memory location that e_1 is evaluated to.

The premise $\ell \le \tau$ is a superficial constraint, which is based on the intuition that it is unnecessary to encrypt a value with a key that is more confidential than the value itself. This constraint is introduced to simplify the proof of noninterference. It does not limit the expressiveness of Sweb because we can always assign a low-confidentiality value to a high-confidentiality location and then encrypt it using a high-confidentiality keystore.

The encryption statement has label $\texttt{stmt}_{\ell \sqcap \ell'}$ because both a memory location of label ℓ' and a keystore of label ℓ are updated by this statement. This labeling prevents illegal implicit flows arising from encryption. For example, consider the following code:

$$\texttt{if}\ !m_s\ \texttt{then}\ m_p := \mathit{encrypt}(m_s,\ K_s)\ \texttt{else skip}$$

where the contents of m_s and K_s are secret, and the value of m_p is public. Because of the encryption, attackers cannot infer the exact value of m_s from the value of m_p after executing the code, but they are able to infer whether the value of m_s is positive. This statement is not well-typed because $m_p := \mathit{encrypt}(m_s,\ K_s)$ has type $\texttt{stmt}_{\ell_p}$, and $!m_s$ has label ℓ_s, and $\ell_s \not\le \ell_p$. The following code demonstrates the implicit flow related to updating the keystore:

$$\texttt{if}\ !m_s\ \texttt{then}\ m_{es} := \mathit{encrypt}(m_s,\ K_p)\ \texttt{else skip}$$

where the value of m_{es} is a secret, but the content of K_p is public. Therefore, attackers can infer whether m_s is positive from how many keys in K_p are used. Again, this statement is not well-typed because $m_{es} := \mathit{encrypt}(m_s,\ K_p)$ has type $\texttt{stmt}_{\ell_p}$.

Consider the web album example discussed in Section 1. The following Sweb code implements storing an encrypted photo (using keystore $K_{\texttt{alice@gmail.com}}$) on $\texttt{album.com}$:

$$m_{\texttt{album.com/ephoto}} := \mathit{encrypt}(!m_{\texttt{photo}},\ K_{\texttt{alice@gmail.com}})$$

Suppose Alice trusts that $\texttt{gmail.com}$ and $\texttt{album.com}$ will not collude to leak her photo, but does not want $\texttt{gmail.com}$ to be able to access her photo. Then the value of $m_{\texttt{photo}}$ has a label ℓ such that $\ell \le \ell_{\texttt{gmail.com}} \sqcup \ell_{\texttt{album.com}}$ and $\ell \not\le \ell_{\texttt{gmail.com}}$. By rule (ENC), the above code is well-typed. However, the code would not be well-typed if the encryption result is treated as public data (with label $\bot$) as in previous work [4, 24].

Rule (SUB) is standard for subtyping. If term t (expression or statement) has type τ, and τ is a subtype of τ', then t has type τ'.

The subtyping rules of Sweb are shown below:

$$\frac{\ell_1 \leq \ell_2}{\beta_{\ell_1} \leq \beta_{\ell_2}} \qquad \frac{\ell_2 \leq \ell_1}{\mathtt{stmt}_{\ell_1} \leq \mathtt{stmt}_{\ell_2}}$$

Intuitively, it is safe to treat low-confidentiality data as high-confidentiality data, and a statement with only high-confidentiality side effects as one with low-confidentiality side effects.

The type system of Sweb satisfies subject reduction. The proof is standard and subsumed by the noninterference proof in Appendix A, so we simply state the theorem here.

Definition 5.1 ($\Gamma ; W \vdash v : \tau$)**.** Value v has type τ with respect to Γ and W, if $\Gamma \vdash v : \tau$, and $\tau = [\tau']_\ell$ implies that $\langle \mathit{decrypt}(v), W\rangle \Downarrow v'$ and $\Gamma ; W \vdash v' : \tau'$.

Definition 5.2 ($\Gamma \vdash W$)**.** W is well-typed with respect to Γ, written as $\Gamma \vdash W$, if $\mathit{dom}(\Gamma) = \mathit{dom}(W)$, and for any m in $\mathit{dom}(\Gamma)$, $\Gamma ; W \vdash W(m) : \Gamma(m)$, and for any f in $\mathit{dom}(\Gamma)$, $\Gamma \vdash W(f) : \Gamma(f)$.

Theorem 5.1 (Subject reduction)**.** Suppose $\Gamma \vdash W$. If $\Gamma \vdash e : \tau$ and $\langle e, W\rangle \Downarrow v$, then $\Gamma \vdash v : \tau$. If $\Gamma \vdash s : \tau$ and $\langle s, W\rangle \longmapsto \langle s', W'\rangle$, then $\Gamma \vdash s' : \tau$ and $\Gamma \vdash W'$.

5.1 Noninterference theorem

Suppose s is a program, and W is the initial web state. The output of s is the trace of web states generated from evaluating $\langle s, W\rangle$. For example, the evaluation $\langle s, W\rangle \longmapsto \langle s_1, W_1\rangle \longmapsto \ldots \longmapsto \langle s_n, W_n\rangle$ generates the trace $T = [W, W_1, \ldots, W_n]$.

The two executions $\langle s, W_1\rangle$ and $\langle s, W_2\rangle$ are indistinguishable to low users if any two traces T_1 and T_2 generated from evaluating the two configurations are low-equivalent. Based on definition 4.1, we can define trace low equivalence, which formalizes the notion of low-equivalent outputs. Intuitively, two traces are low-equivalent if they may be generated by the same execution (one trace appears to be the prefix of the other) from the perspective of low users. Formally, the low-equivalence relation between two traces is defined as follows (where notation $T_1 \approx T_2$ means that T_1 and T_2 are equal up to stuttering):

Definition 5.3 ($\Gamma \vdash T_1 \approx_L T_2$)**.** There exist $T_1' = [W_1, \ldots, W_n]$ and $T_2' = [W_1', \ldots, W_m']$ such that $T_1 \approx T_1'$, and $T_2 \approx T_2'$, and $\Gamma \vdash W_i \approx_L W_i'$ for any i in $\{1, \ldots, \mathtt{min}(m, n)\}$.

With the notion of low-equivalent traces, it is straightforward to define the noninterference theorem:

Theorem 5.2 (Noninterference)**.** Suppose $\Gamma \vdash s : \tau$, and $\Gamma \vdash W_1 \approx_L W_2$. If T_1 and T_2 are the two traces of evaluating $\langle s, W_1\rangle$ and $\langle s, W_2\rangle$, respectively, then $\Gamma \vdash T_1 \approx_L T_2$.

Proof. See Appendix A. □

6. Related work

Using static program analysis to check information flow was first proposed by Denning and Denning [10]; later work phrased the analysis as type checking (e.g., [18]). Noninterference was later developed as a more semantic characterization of security [11], followed by many extensions. Volpano, Smith and Irvine [26] first showed that type systems can be used to enforce noninterference, and proved a version of noninterference theorem for a simple imperative language, starting a line of research pursuing the noninterference result for more expressive security-typed languages [12, 28, 6, 19].

More recent work looked into security-typed languages with cryptographic primitives. Laud and Vene [14] presented a type system for enforcing computationally secure information flow in the presence of encryption. Askarov, Hedin and Sabelfeld [4] studied a language with encryption, decryption and key generation primitives, and showed its type system enforces possibilistic noninterference. In comparison, our work considers a rather distinctive set of cryptographic primitives that do not manipulate keys explicitly. Moreover, the type systems in those previous work treat encryption results as public data, and the treatment is too restrictive to handle the case that an encryption key is less confidential than the plaintext it encrypts. In contrast, the type system of Sweb assigns label to a ciphertext based on the label of the encryption key, leading to more permissive typing. The work of Askarov, Hedin and Sabelfeld used possibilistic noninterference to avoid masking implicit flows in ciphertexts. In our work, this issue is dealt with by considering the preservation of equality relation between corresponding ciphertexts.

Other work studied more abstract cryptography-related primitives. Smith and Alpízar [24] investigated a random assignment operator and showed a security-typed language with this operator enforces probabilistic noninterference. Their work also considered the encryption and decryption primitives, but also had the limitation of assigning the lowest label to encryption results. Vaughan and Zdancewic [25] considered abstract packaging operators that rely on both static and dynamic checking for information flow control.

Abadi [1] presented a basic concurrent language (the spi calculus) with cryptographic primitives and a type system for enforcing secrecy. Rather than modeling an information flow analysis, the typing rules of the spi calculus formalize the principles and rules for achieving secrecy properties in security protocols.

Also related is work on connecting formal cryptographic analysis techniques and computational security models. For example, Abadi and Rogaway [2] proved the computational soundness of Dolev-Yao analysis. More recently, Backes and Pfitzmann [5] investigated a Dolev-Yao style cryptographic library and established the relation between symbolic and cryptographic secrecy properties for cryptographic protocols.

Jammalamadaka et al. [13] presented the gVault system, a cryptographic network file system built on the Gmail service. In gVault, encryption keys are generated and recomputed using user passwords, which is susceptible to dictionary attacks and requires a password recovery mechanism that may have usability issues. Moreover, it is not clear that the password-based key management can be applied to more complex web applications involving multiple sites.

Declassification constructs have been introduced in a few security-typed languages [17, 15] for intentional information releases. A typical use of these constructs is to release encryption results of confidential data to low users. However, a declassification mechanism is generally too powerful to allow any noninterference-like assertion being made.

The sequential programming model for distributed systems with untrusted components was first used in the secure program partitioning work [30, 31] and later in the Swift system [8]. We use this model for its simplicity rather than because it makes programming distributed applications easier.

7. Conclusions

This paper presents a nonintrusive encryption mechanism for the Web. The core idea is to make key generation and management transparent, to achieve high usability. Although it prevents key reuse, transparent key management is practical for the Web environment since large number of encryption keys can be easily stored on the Web. This paper also proves the soundness of the encryption mechanism in the context of a security-typed language, which provides a permissive and flexible way of typing the encryption primitive, formalizing the observation that the confidentiality of a

plaintext can be protected by keeping either the ciphertext or the encryption key confidential.

In Sweb, each encryption is assumed to take place with a new key. At the cost of a more complex dependent type system, one could imagine separating key generation from encryption, which would allow Sweb to be used to describe more sophisticated protocols. This is worth further investigation.

Acknowledgements

The authors would like to thank Michael Clarkson for his insightful suggestions and comments on this work. Thanks also to the anonymous reviewers for their helpful feedback.

This work was supposed by the National Science Foundation under grants 0430161 and 0627649. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of the NSF or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

[1] Martín Abadi. Secrecy by typing in security protocols. In *Proc. Theoretical Aspects of Computer Software: Third International Conference*, September 1997.

[2] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *TCS '00: Proceedings of the International Conference IFIP on Theoretical Computer Science*, pages 3–22, London, UK, 2000.

[3] Johan Agat. Transforming out timing leaks. In *Proc. 27th ACM Symp. on Principles of Programming Languages (POPL)*, pages 40–53, Boston, MA, January 2000.

[4] Aslan Askarov, Daniel Hedin, and Andrei Sabelfeld. Cryptographically-masked flows. In *Proc. 13th International Static Analysis Symposium*, Seoul, Korea, August 2006.

[5] Michael Backes and Birgit Pfitzmann. Relating symbolic and cryptographic secrecy. *IEEE Trans. Dependable Secur. Comput.*, 2(2):109–123, 2005.

[6] Anindya Banerjee and David A. Naumann. Secure information flow and pointer confinement in a Java-like language. In *Proc. 15th IEEE Computer Security Foundations Workshop*, June 2002.

[7] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption: Analysis of DES modes of operation. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*, Washington, DC, USA, 1997.

[8] Stephen Chong, Jed Liu, Andrew C. Myers, Xin Qi, K. Vikram, Lantian Zheng, and Xin Zheng. Secure web applications via automatic partitioning. In *Proc. 21st ACM Symp. on Operating System Principles (SOSP)*, October 2007.

[9] Dorothy E. Denning. A lattice model of secure information flow. *Comm. of the ACM*, 19(5):236–243, 1976.

[10] Dorothy E. Denning and Peter J. Denning. Certification of programs for secure information flow. *Comm. of the ACM*, 20(7):504–513, July 1977.

[11] Joseph A. Goguen and Jose Meseguer. Security policies and security models. In *Proc. IEEE Symposium on Security and Privacy*, pages 11–20, April 1982.

[12] Nevin Heintze and Jon G. Riecke. The SLam calculus: Programming with secrecy and integrity. In *Proc. 25th ACM Symp. on Principles of Programming Languages (POPL)*, pages 365–377, San Diego, California, January 1998.

[13] Ravi Chandra Jammalamadaka, Roberto Gamboni, Sharad Mehrotra, Kent E. Seamons, and Nalini Venkatasubramanian. gvault: A gmail based cryptographic network file system. In *Proceedings of 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, pages 161–176, 2007.

[14] Peeter Laud and Varmo Vene. A type system for computationally secure information flow. In *Proceedings of the 15th International Symposium on Fundamentals of Computational Theory*, pages 365–377, Lübeck, Germany, 2005.

[15] Peng Li and Steve Zdancewic. Downgrading policies and relaxed noninterference. In *Proc. 32nd ACM Symp. on Principles of Programming Languages (POPL)*, Long Beach, CA, January 2005.

[16] Andrew C. Myers and Barbara Liskov. A decentralized model for information flow control. In *Proc. 17th ACM Symp. on Operating System Principles (SOSP)*, pages 129–142, Saint-Malo, France, 1997.

[17] Andrew C. Myers, Lantian Zheng, Steve Zdancewic, Stephen Chong, and Nathaniel Nystrom. Jif: Java information flow. Software release, `http://www.cs.cornell.edu/jif`, July 2001.

[18] Jens Palsberg and Peter Ørbæk. Trust in the λ-calculus. In *Proc. 2nd International Symposium on Static Analysis*, number 983 in Lecture Notes in Computer Science, pages 314–329. Springer, September 1995.

[19] François Pottier and Vincent Simonet. Information flow inference for ML. In *Proc. 29th ACM Symp. on Principles of Programming Languages (POPL)*, pages 319–330, 2002.

[20] Andrei Sabelfeld and Heiko Mantel. Static confidentiality enforcement for distributed programs. In *Proc. 9th International Static Analysis Symposium*, volume 2477 of *LNCS*, Madrid, Spain, September 2002. Springer-Verlag.

[21] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, January 2003.

[22] Bruce Schneier. *Applied Cryptography*. John Wiley and Sons, New York, NY, 1996.

[23] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[24] Geoffrey Smith and Rafael Alpízar. Secure information flow with random assignment and encryption. In *FMSE '06: Proceedings of the fourth ACM workshop on Formal methods in security*, pages 33–44, Alexandria, Virginia, USA, 2006.

[25] Jeffrey A. Vaughan and Steve Zdancewic. A cryptographic decentralized label model. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pages 192–206, May 2007.

[26] Dennis Volpano, Geoffrey Smith, and Cynthia Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.

[27] Steve Zdancewic and Andrew C. Myers. Robust declassification. In *Proc. 14th IEEE Computer Security Foundations Workshop*, pages 15–23, June 2001.

[28] Steve Zdancewic and Andrew C. Myers. Secure information flow via linear continuations. *Higher Order and Symbolic Computation*, 15(2–3):209–234, September 2002.

[29] Steve Zdancewic and Andrew C. Myers. Observational determinism for concurrent program security. In *Proc. 16th IEEE Computer Security Foundations Workshop*, pages 29–43, Pacific Grove, California, June 2003.

[30] Steve Zdancewic, Lantian Zheng, Nathaniel Nystrom, and Andrew C. Myers. Secure program partitioning. *ACM Transactions on Computer Systems*, 20(3):283–328, August 2002.

[31] Lantian Zheng, Stephen Chong, Andrew C. Myers, and Steve Zdancewic. Using replication and partitioning to build secure distributed systems. In *Proc. IEEE Symposium on Security and Privacy*, pages 236–250, Oakland, California, May 2003.

[32] Lantian Zheng and Andrew C. Myers. End-to-end availability policies and noninterference. In *Proc. 18th IEEE Computer Security Foundations Workshop*, pages 272–286, June 2005.

A. Noninterference proof

The noninterference result for Sweb is proved by extending the language to a new language XSweb. Each configuration C in XSweb encodes two Sweb configurations C_1 and C_2. Moreover, the operational semantics of XSweb is consistent with that of Sweb in the sense that the result of evaluating C is an encoding of the results of evaluating C_1 and C_2 in Sweb. The type system of XSweb can guarantee that C is well-typed only if the low-confidentiality parts of C_1 and C_2 are equivalent. Intuitively, if the result of C is well-typed, then the results of evaluating C_1 and C_2 should also have equivalent low-confidentiality parts. Therefore, the preservation of type soundness in an XSweb evaluation implies the preservation of low-equivalence between two Sweb evaluations. Thus, to prove the noninterference theorem of Sweb, we only need to prove the subject reduction theorem of XSweb. This proof technique was first used by Pottier and Simonet to prove the noninterference result of a security-typed ML-like language [19].

A.1 Syntax extensions

The syntax extensions of XSweb include the bracket constructs, which are composed of two Sweb terms and used to capture the differences between two Sweb configurations.

$$\begin{array}{rrcl} \text{Values} & v & ::= & \ldots \mid (v_1 \mid v_2) \\ \text{Statements} & s & ::= & \ldots \mid (s_1 \mid s_2) \end{array}$$

The bracket constructs cannot be nested, so the subterms of a bracket construct must be Sweb terms. Given an XSweb statement s, let $\lfloor s \rfloor_1$ and $\lfloor s \rfloor_2$ represent the two Sweb statements that s encodes. The projection functions satisfy $\lfloor (s_1 \mid s_2) \rfloor_i = s_i$ and are homomorphisms on other statement and expression forms. An XSweb state W maps references to XSweb terms that encode two Sweb terms. Thus, the projection function can be defined on web states too. For $i \in \{1, 2\}$, $\mathit{dom}(\lfloor W \rfloor_i) = \mathit{dom}(W)$, and for any $m \in \mathit{dom}(W)$, $\lfloor W \rfloor_i(m) = \lfloor W(m) \rfloor_i$.

Since an XSweb term effectively encodes two Sweb terms, the evaluation of a XSweb term can be projected into two Sweb evaluations. An evaluation step of a bracket statement $(s_1 \mid s_2)$ is an evaluation step of either s_1 or s_2, and s_1 or s_2 can only access the corresponding projection of the web state. Thus, the configuration of XSweb has an index $i \in \{\bullet, 1, 2\}$ that indicates whether the term to be evaluated is a subterm of a bracket expression, and if so, which branch of a bracket the term belongs to. For example, the configuration $\langle s,\, W \rangle_1$ means that s belongs to the first branch of a bracket, and s can only access the first projection of W. We write "$\langle s,\, W \rangle$" for "$\langle s,\, W \rangle_\bullet$", which means s does not belong to any bracket.

The operational semantics of XSweb is shown in Figure 4. It is based on the semantics of Sweb and contains some new evaluation rules (E5), (S8–S11) for manipulating bracket constructs. Rules (E1), (S6) and (S7) are modified to access the web state projection corresponding to index i. The rest of the rules in Figure 2 are adapted to XSweb by indexing each configuration with i. The following adequacy and soundness lemmas state that the operational semantics of XSweb is adequate to encode the execution of two Sweb terms.

Let the notation $\langle s,\, W \rangle \longmapsto^T \langle s',\, W' \rangle$ denote that $\langle s,\, W \rangle \longmapsto \langle s_1,\, W_1 \rangle \longmapsto \ldots \longmapsto \langle s_n,\, W_n \rangle \longmapsto \langle s',\, W' \rangle$ and $T = [W, W_1, \ldots, W_n, W']$, or $s = s'$ and $W = W'$ and $T = [W]$. In addition, let $|T|$ denote the length of T, and $T_1 \oplus T_2$ denote the trace obtained by concatenating T_1 and T_2. Suppose $T_1 = [W_1, \ldots, W_n]$ and $T_2 = [W'_1, \ldots, W'_m]$. If $W_n = W'_1$, then $T_1 \oplus T_2 = [W_1, \ldots, W_n, W'_2, \ldots, W'_m]$. Otherwise, $T_1 \oplus T_2 = [W_1, \ldots, W_n, W'_1, \ldots, W'_m]$.

Lemma A.1 (Projection i)**.** Suppose $\langle e,\, W \rangle \Downarrow v$. Then for $i \in \{1, 2\}$, $\langle \lfloor e \rfloor_i,\, \lfloor W \rfloor_i \rangle \Downarrow \lfloor v \rfloor_i$ holds.

$$\text{(E1)}\quad \frac{\pi_i\, W(m) = v \qquad v \neq \texttt{none}}{\langle !m,\, W \rangle_i \Downarrow v}$$

$$\text{(E4)}\quad \frac{\langle e_1,\, W \rangle_i \Downarrow v_1 \qquad \langle e_2,\, W \rangle_i \Downarrow v_2 \qquad v = v_1 \oplus v_2}{\langle e_1 + e_2,\, W \rangle \Downarrow v}$$

$$\text{(E5)}\quad \frac{\begin{array}{c}\langle e,\, W \rangle \Downarrow v \quad \lfloor v \rfloor_1 \neq \lfloor v \rfloor_2 \\ \langle \mathit{decrypt}(\lfloor v \rfloor_i),\, W \rangle_i \Downarrow v_i,\ i \in \{1, 2\}\end{array}}{\langle \mathit{decrypt}(e),\, W \rangle \Downarrow (v_1 \mid v_2)}$$

$$\text{(S6)}\quad \frac{\begin{array}{c}\langle e_1,\, W \rangle_i \Downarrow m \quad \langle e_2,\, W \rangle_i \Downarrow v \quad W(K) = \mathcal{K} \\ \mathit{newkey}(\lfloor \mathcal{K} \rfloor_i) = \texttt{i}{:}k \quad \mathcal{E}(v,\, k) = c \\ W'' = W[m \mapsto W(m)[c.K.\texttt{i}/\pi_i]] \\ W' = W[K \mapsto \mathcal{K}[\texttt{i} \mapsto_i k]]\end{array}}{\langle e_1 := \mathit{encrypt}(e_2,\, K),\, W \rangle_i \longmapsto \langle \texttt{skip},\, W' \rangle_i}$$

$$\text{(S7)}\quad \frac{\langle e_1,\, W \rangle_i \Downarrow m \quad \langle e_2,\, W \rangle_i \Downarrow v}{\langle e_1 := e_2,\, W \rangle_i \longmapsto \langle \texttt{skip},\, W[m \mapsto W(m)[v/\pi_i]] \rangle_i}$$

$$\text{(S8)}\quad \frac{\langle e,\, W \rangle \Downarrow (n_1 \mid n_2)}{\begin{array}{l}\langle \texttt{if}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2,\, W \rangle \longmapsto \\ \quad \langle (\texttt{if}\ n_1\ \texttt{then}\ \lfloor s_1 \rfloor_1\ \texttt{else}\ \lfloor s_2 \rfloor_1 \ \mid \\ \quad\ \ \texttt{if}\ n_2\ \texttt{then}\ \lfloor s_1 \rfloor_2\ \texttt{else}\ \lfloor s_2 \rfloor_2),\, W \rangle\end{array}}$$

$$\text{(S9)}\quad \frac{\langle s_i,\, W \rangle_i \longmapsto \langle s'_i,\, W' \rangle_i \quad s_j = s'_j \quad \{i, j\} = \{1, 2\}}{\langle (s_1 \mid s_2),\, W \rangle \longmapsto \langle (s'_1 \mid s'_2),\, W' \rangle}$$

$$\text{(S10)}\quad \langle (\texttt{skip} \mid \texttt{skip}),\, W \rangle \longmapsto \langle \texttt{skip},\, W \rangle$$

$$\text{(S11)}\quad \frac{\langle e_1,\, W \rangle \Downarrow (m_1 \mid m_2)}{\langle e_1 := e_2,\, W \rangle \longmapsto \langle (m_1 := \lfloor e_2 \rfloor_1 \mid m_2 := \lfloor e_2 \rfloor_2),\, W \rangle}$$

$$\text{(S12)}\quad \frac{\begin{array}{c}\langle e_1,\, W \rangle \Downarrow (m_1 \mid m_2) \\ \text{Let } s_i \text{ be } m_1 := \mathit{encrypt}(\lfloor e_2 \rfloor_1,\, K),\ i \in \{1, 2\}\end{array}}{\langle e_1 := \mathit{encrypt}(e_2,\, K),\, W \rangle \longmapsto \langle (s_1 \mid s_2),\, W \rangle}$$

[Auxiliary functions]

$$\begin{array}{ll} v[v'/\pi_\bullet] = v' & \pi_\bullet\, v = v \\ v[v'/\pi_1] = (v' \mid \lfloor v \rfloor_2) & \pi_1\, v = \lfloor v \rfloor_1 \\ v[v'/\pi_2] = (\lfloor v \rfloor_1 \mid v') & \pi_2\, v = \lfloor v \rfloor_2 \\ v[(c_1 \mid c_2).K.(\texttt{i}_1 \mid \texttt{i}_2)/\pi_\bullet] = (c_1.K.\texttt{i}_1 \mid c_2.K.\texttt{i}_2) & \end{array}$$

Figure 4. The operational semantics of XSweb

Proof. By induction on the structure of e. □

Lemma A.2 (Projection ii)**.** Suppose W is an XSweb state, and $\lfloor W \rfloor_i = W_i$ for $i \in \{1, 2\}$, and $\langle s,\, W_i \rangle$ is an Sweb configuration. Then $\langle s,\, W_i \rangle \longmapsto \langle s',\, W'_i \rangle$ if and only if $\langle s,\, W \rangle_i \longmapsto \langle s',\, W' \rangle_i$ and $\lfloor W' \rfloor_i = W'_i$.

Proof. By induction on the structure of s. □

Lemma A.3 (Expression adequacy)**.** If for $i \in \{1, 2\}$, $\langle e_i,\, W_i \rangle \Downarrow v_i$, and there exists $\langle e,\, W \rangle$ in XSweb such that $\lfloor e \rfloor_i = e_i$ and $\lfloor W \rfloor_i = W_i$. Then $\langle e,\, W \rangle \Downarrow v$ such that $\lfloor v \rfloor_i = v_i$.

Proof. By induction on the structure of e. □

Lemma A.4 (One-step adequacy)**.** Suppose for $i \in \{1, 2\}$, $\langle s_i,\, W_i \rangle \longmapsto \langle s'_i,\, W'_i \rangle$ is an evaluation in Sweb, and there exists $\langle s,\, W \rangle$ in XSweb such that $\lfloor s \rfloor_i = s_i$ and $\lfloor W \rfloor_i = W_i$. Then there exists $\langle s',\, W' \rangle$ such that $\langle s,\, W \rangle \longmapsto^T \langle s',\, W' \rangle$, and one of the following conditions holds:

i. For $i \in \{1,2\}$, $\lfloor T \rfloor_i \approx [W_i, W_i']$ and $\lfloor s' \rfloor_i = s_i'$.
ii. For $\{j,k\} = \{1,2\}$, $\lfloor T \rfloor_j \approx [W_j]$ and $\lfloor s' \rfloor_j = s_j$, and $\lfloor T \rfloor_k \approx [W_k, W_k']$ and $\lfloor s' \rfloor_k = s_k'$.

Proof. By induction on the structure of s. The proof is largely similar to the one in the noninterference proof of Aimp [32]. We just show some cases here.

- s is $e_1 := e_2$. In this case, s_i is $\lfloor e_1 \rfloor_i := \lfloor e_2 \rfloor_i$, and $\langle \lfloor e_1 \rfloor_i := \lfloor e_2 \rfloor_i, W_i \rangle \longmapsto \langle \texttt{skip}, W_i[m_i \mapsto v_i] \rangle$ where $\langle \lfloor e_1 \rfloor_i, W_i \rangle \Downarrow m_i$ and $\langle \lfloor e \rfloor_i, W_i \rangle \Downarrow v_i$. By Lemma A.3, we have $\langle e_1, W \rangle \Downarrow m$ such that $\lfloor m \rfloor_i = m_i$, and $\langle e_2, W \rangle \Downarrow v$ such that $\lfloor v \rfloor_i = v_i$. If $m_1 = m_2$, then $\langle e_1 := e_2, W \rangle \longmapsto \langle \texttt{skip}, W[m \mapsto v] \rangle$. Since $\lfloor W \rfloor_i = W_i$, we have $\lfloor W[m \mapsto v] \rfloor_i = W_i[m \mapsto \lfloor v \rfloor_i]$. Finally, we have $\lfloor s' \rfloor_i = s_i' = \texttt{skip}$ for $i \in \{1,2\}$. If $m_1 \neq m_2$, then $\langle s, W \rangle \longmapsto \langle (\lfloor e_1 \rfloor_1 := \lfloor e_2 \rfloor_1 \mid \lfloor e_1 \rfloor_2 := \lfloor e_2 \rfloor_2), W \rangle \longmapsto \langle (\texttt{skip} \mid \lfloor e_1 \rfloor_2 := \lfloor e_2 \rfloor_2), W[m_1 \mapsto W(m_1)[v_1/\pi_1]] \rangle$. It is easy to verify that this execution satisfies condition (ii).
- s is $e_1 := \mathit{encrypt}(e_2, K)$. By the same argument as the above case.
- s is $\texttt{call}\ f$. Then s_i is also $\texttt{call}\ f$, and $\langle s_i, W_i \rangle \longmapsto \langle s', W_i \rangle$ where $s' = W_i(f)$. Therefore, $\langle s, W \rangle \longmapsto \langle s, W \rangle$.

□

Lemma A.5 (Adequacy)**.** Suppose $\langle s_i, W_i \rangle \longmapsto^{T_i} \langle s_i', W_i' \rangle$ for $i \in \{1,2\}$ are two evaluations in Sweb. Then for an XSweb configuration $\langle s, W \rangle$ such that $\lfloor s \rfloor_i = s_i$ and $\lfloor W \rfloor_i = W_i$ for $i \in \{1,2\}$, we have $\langle s, W \rangle \longmapsto^{T} \langle s', W' \rangle$ such that $\lfloor T \rfloor_j \approx T_j$ and $\lfloor T \rfloor_k \approx T_k'$, where T_k' is a prefix of T_k and $\{k,j\} = \{1,2\}$.

Proof. By induction on the sum of the lengths of T_1 and T_2: $|T_1| + |T_2|$.

- $|T_1| + |T_2| \leq 3$. Without loss of generality, suppose $|T_1| = 1$. Then $T_1 = [W_1]$. Let $T = [W]$. We have $\langle s, W \rangle \longmapsto^{T} \langle s, W \rangle$. It is clear that $\lfloor T \rfloor_1 = T_1$, and $\lfloor T \rfloor_2 = [W_2]$ is a prefix of T_2.
- $|T_1| + |T_2| > 3$. If $|T_1| = 1$ or $|T_2| = 1$, then the same argument in the above case applies. Otherwise, we have $\langle s_i, W_i \rangle \longmapsto \langle s_i'', W_i'' \rangle \longmapsto^{T_i'} \langle s_i', W_i' \rangle$ and $T_i = [W_i] \oplus T_i'$ for $i \in \{1,2\}$. By Lemma A.4, $\langle s, W \rangle \longmapsto^{T'} \langle s'', W'' \rangle$ such that
  i. For $i \in \{1,2\}$, $\lfloor T' \rfloor_i \approx [W_i, W_i'']$ and $\lfloor s'' \rfloor_i = s_i''$. Since $|T_1'| + |T_2'| < |T_1| + |T_2|$, by induction we have $\langle s'', W'' \rangle \longmapsto^{T''} \langle s', W' \rangle$ such that for $\{k,j\} = \{1,2\}$, $\lfloor T'' \rfloor_j \approx T_j'$ and $\lfloor T'' \rfloor_k \approx T_k''$, and T_k'' is a prefix of T_k'. Let $T = T' \oplus T''$. Then $\langle s, W \rangle \longmapsto^{T} \langle s', W' \rangle$, and $\lfloor T \rfloor_j \approx T_j$, and $\lfloor T \rfloor_k \approx T_k'$ where $T_k' = [W_k, W_k''] \oplus T_k''$ is a prefix of T_k.
  ii. For $\{j,k\} = \{1,2\}$, $\lfloor T' \rfloor_j \approx [W_j]$ and $\lfloor s \rfloor_j = s_j$, and $\lfloor T' \rfloor_k \approx [W_k, W_k'']$ and $\lfloor s \rfloor_k = s_k''$. Without loss of generality, suppose $j = 1$ and $k = 2$. Since $\langle s_1, W_1 \rangle \longmapsto^{T_1} \langle s_1', W_1' \rangle$ and $\langle s_2'', W'' \rangle \longmapsto^{T_2'} \langle s_2', W_2' \rangle$, and $\lfloor s' \rfloor_1 = s_1$ and $\lfloor s' \rfloor_2 = s_2''$, and $|T_2'| < |T_2|$, we can apply the induction hypothesis to $\langle s'', W'' \rangle$. By the similar argument in the above case, this lemma holds for this case.

□

Lemma A.6 (Soundness)**.** Suppose $\langle s, W \rangle \longmapsto \langle s', W' \rangle$. Then $\langle \lfloor s \rfloor_i, \lfloor W \rfloor_i \rangle \longmapsto^* \langle \lfloor s' \rfloor_i, \lfloor W' \rfloor_i \rangle$.

Proof. By induction on the derivation of $\langle s, W \rangle \longmapsto \langle s', W' \rangle$. □

A.2 Typing rules

The type system of XSweb includes all the typing rules in Figure 3 and has two additional rules for typing bracket constructs. The bracket constructs captures the differences between two Sweb configurations. As a result, any effect and result of a bracket construct should have a high label ℓ ($\ell \not\leq L$) except for a bracket of two encrypted values. Consider a bracket $(v_1 \mid v_2)$ with type $[\tau]_\ell$. If $\ell \leq L$ and $\tau \not\leq L$, then low users still cannot differentiate the two executions from the value. Type τ itself may be an encrypted type $[\tau']_{\ell'}$. Then ℓ' may be low if τ' has a high label. Let notation $\mathit{label}^+(\tau)$ be ℓ if $\tau = \beta_\ell$ and β is not $[\tau']$, or $\ell \sqcup \mathit{label}^+(\tau')$ if $\tau = [\tau']_\ell$. Then a bracket value $(v_1 \mid v_2)$ has type τ if both v_1 and v_2 have type τ and $\mathit{label}^+(\tau) \not\leq L$.

$$\text{(V-PAIR)} \quad \frac{\Gamma \vdash v_1 : \tau \quad \Gamma \vdash v_2 : \tau \quad \mathit{label}^+(\tau) \not\leq L}{\Gamma \vdash (v_1 \mid v_2) : \tau}$$

$$\text{(S-PAIR)} \quad \frac{\Gamma \vdash s_1 : \tau \quad \Gamma \vdash s_2 : \tau \quad \tau \not\leq L}{\Gamma \vdash (s_1 \mid s_2) : \tau}$$

A.3 Subject reduction

Lemma A.7 (Update)**.** Suppose $\Gamma \vdash v : \tau$, and $\Gamma \vdash v' : \tau$, and $i \in \{1,2\}$ implies that $\tau \not\leq L$. Then $\Gamma \vdash v[v'/\pi_i] : \tau$.

Proof. If i is $\bullet$, then $v[v'/\pi_i] = v'$, and we have $\Gamma \vdash v' : \tau$. If i is 1, then $v[v'/\pi_i] = (v' \mid \lfloor v \rfloor_2)$ and $\tau \not\leq L$. Since $\Gamma \vdash v : \tau$, we have $\Gamma \vdash \lfloor v \rfloor_2 : \tau$. By rule (V-PAIR), $\Gamma \vdash (v' \mid \lfloor v \rfloor_2) : \tau$. Similarly, if i is 2, we also have $\Gamma \vdash v[v'/\pi_i] : \tau$. □

Definition A.1 ($\Gamma \vdash W$)**.** W is well-typed with respect to Γ, written $\Gamma \vdash W$, if $\mathit{dom}(\Gamma) = \mathit{dom}(W)$ and the following conditions hold:

- $\forall m \in \mathit{dom}(\Gamma).\ \Gamma; W \vdash W(m) : \Gamma(m)$.
- For any f, $\Gamma \vdash W(f) : \Gamma(f)$.
- For any K, if $\mathit{label}(K) \leq L$, then $\lfloor W(K) \rfloor_1 = \lfloor W(K) \rfloor_2$.
- For any m_1, m_2 such that $\mathit{label}(m_1) \sqcup \mathit{label}(m_2) \leq L$, $\lfloor W \rfloor_1^{i,L}(m_1) = \lfloor W \rfloor_1^{j,L}(m_2)$ iff $\lfloor W \rfloor_2^{i,L}(m_1) = \lfloor W \rfloor_2^{j,L}(m_2)$.

Lemma A.8. Suppose $\Gamma \vdash e : \tau$, and $\Gamma \vdash W$, and $\langle e, W \rangle \Downarrow v$. Then $\Gamma \vdash v : \tau$.

Proof. By induction on the structure of e. □

Lemma A.9. Suppose $\Gamma \vdash W$, and $\Gamma \vdash e : \tau$ such that $\tau \leq L$. If $\langle e, W \rangle \Downarrow (v_1 \mid v_2)$, then for any m such that $\mathit{label}(m) \leq L$, $\lfloor W \rfloor_1^{i,L}(m) = \lfloor W \rfloor_1^{j,L}(v_1)$ iff $\lfloor W \rfloor_2^{i,L}(m) = \lfloor W \rfloor_2^{j,L}(v_2)$.

Proof. By induction on the structure of e. □

Theorem A.1 (Subject reduction)**.** Suppose $\Gamma \vdash s : \tau$, and $\Gamma \vdash W$, and $\langle s, W \rangle_i \longmapsto \langle s', W' \rangle_i$, and $i \in \{1,2\}$ implies $\tau \not\leq L$. Then $\Gamma \vdash s' : \tau$ and $\Gamma \vdash W'$.

Proof. By induction on the evaluation step $\langle s, W \rangle_i \longmapsto \langle s', W' \rangle_i$. The cases for rules (S5) and (S10) are trivial.

- **Case (S1)**. In this case, s is $\texttt{if}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2$. By the typing rule (IF), we have $\Gamma \vdash s_1 : \tau$.
- **Case (S2)**. By the same argument as case (S1).
- **Case (S3)**. In this case, s is $\texttt{call}\ f$, and s' is $W(f)$. By rule (FUN), $\Gamma(f) = \tau$. Since $\Gamma \vdash W$, we have $\Gamma \vdash s' : \tau$.
- **Case (S4)**. By induction.

- **Case (S6)**. s is $e_1 := \mathit{encrypt}(e_2,\ K)$, and s' is `skip`. So $\Gamma \vdash s' : \tau$ immediately holds. By rule (S6), we have $\langle e_1,\ W\rangle_i \Downarrow m$, and $\langle e_2,\ W\rangle_i \Downarrow v$. If $i \in \{1, 2\}$, then $\tau \not\leq L$, which implies that $\mathit{label}(m) \not\leq L$ and $\mathit{label}(K) \not\leq L$. Therefore, $\Gamma \vdash W'$. Now consider the case that $\mathit{label}(m) \leq L$. Suppose $\Gamma \vdash e_2 : \tau_e$. If $\tau_e \leq L$, then v is not a bracket value, and $\mathit{label}(K) \leq L$. Thus, $(\mathtt{i} : k) = \mathit{newkey}(K)$, and $c = \mathcal{E}(v,\ k)$, and $W'' = W[m \mapsto c.K.\mathtt{i}]$. It is clear that $\Gamma \vdash c.K.\mathtt{i} : [\tau_e]_\ell$, and the decryption result of $W''(m)$ is v, which has type τ_e by Lemma A.8. Therefore, $\Gamma \vdash W''$. Furthermore, since $W' = W''[K \mapsto \mathcal{K}']$, we have $\Gamma \vdash W'$.
  Suppose $\tau_e \not\leq L$, and $\Gamma(m) = [\tau_e]_\ell$. If $\mathit{label}(K) \leq L$, then $\ell \not\leq L$, and we have $\Gamma \vdash W'$. Otherwise, $\mathit{label}(K) \not\leq L$, and $(\mathtt{i}_1 \mid \mathtt{i}_2) : (k_1 \mid k_2) = \mathit{newkey}(K)$. Thus, $(c_1 \mid c_2) = \mathcal{E}(v,\ (k_1 \mid k_2))$. By rule (V-PAIR), $\Gamma \vdash (c_1.K.\mathtt{i}_1 \mid c_2.K.\mathtt{i}_2) : [\tau_e]_\ell$. Since the keys corresponding to $\mathtt{i}_1$ and $\mathtt{i}_2$ are new keys, there does not exist m' and i and j such that $c_i.K.\mathtt{i}_i \neq \lfloor W \rfloor_i^{j,L}(m')$. Therefore, $\Gamma \vdash W'$.
- **Case (S7)**. The interesting scenario is that i is $\bullet$, and e_2 has type $[\tau']_\ell$ such that $\ell \leq L$. Suppose $\langle e_1,\ W\rangle \Downarrow v$, and $v = (v_1 \mid v_2)$. Then $\mathit{label}^+(\tau') \not\leq L$. By Lemma A.9, $\Gamma \vdash W'$.
- **Case (S8)**. In this case, s is $\mathtt{if}\ e \mathtt{then} s_1 \mathtt{else} s_2$, and i must be $\bullet$. Suppose $\Gamma \vdash e : \mathtt{int}_\ell$. By Lemma A.8, $\Gamma \vdash (n_1 | n_2) : \mathtt{int}_\ell$. By rule (V-PAIR), $\ell \not\leq L$. By rule (IF), $\Gamma \vdash s_i : \tau$ for $i \in \{1, 2\}$. Therefore, $\Gamma \vdash \mathtt{if}\ n_i\ \mathtt{then}\ \lfloor s_1 \rfloor_i\ \mathtt{else}\ \lfloor s_2 \rfloor_i : \tau$ for $i \in \{1, 2\}$. By rule (S-PAIR), $\Gamma \vdash s' : \tau$, because $\tau \not\leq L$.
- **Case (S9)**. In this case, s is $(s_1 \mid s_2)$. Without loss of generality, suppose $\langle s_1,\ W\rangle_1 \longmapsto \langle s_1',\ W'\rangle_1$, and $\langle s,\ W\rangle \longmapsto \langle (s_1' \mid s_2),\ W'\rangle$. By rule (S-PAIR), $\Gamma \vdash s_1 : \tau$. By induction, $\Gamma \vdash s_1' : \tau$ and $\Gamma \vdash W'$. By rule (S-PAIR), $\Gamma \vdash s' : \tau$ since $\tau \not\leq L$.
- **Case (S11)**. In this case, $\Gamma \vdash e_1 : \tau'\ \mathtt{ref}_\ell$ and $\ell \not\leq L$, which implies $\tau \not\leq L$. By rule (S-PAIR), $\Gamma \vdash s' : \tau$.
- **Case (S12)**. By the same argument as in case (S11).

□

A.4 Noninterference

Theorem A.2 (Noninterference)**.** If $\Gamma \vdash s : \tau$, then s satisfies the noninterference property.

Proof. Given W_1 and W_2 in Sweb, let $W = W_1 \uplus W_2$ be an XSweb state computed as follows:

$$W_1 \uplus W_2(r) = \begin{cases} W_1(r) & \text{if } W_1(r) = W_2(r) \\ (W_1(r) \mid W_2(r)) & \text{if } W_1(r) \neq W_2(r) \end{cases}$$

Then $\Gamma \vdash W_1 \approx_L W_2$ implies $\Gamma \vdash W$. Suppose $\langle s_i,\ W_i\rangle \longmapsto^{T_i} \langle s_i',\ W'\rangle$ for $i \in \{1, 2\}$. Then by Lemma A.5, there exists $\langle s',\ W'\rangle$ such that $\langle s,\ W\rangle \longmapsto^{T} \langle s',\ W'\rangle$, and $\lfloor T \rfloor_j \approx T_j$ and $\lfloor T \rfloor_k \approx T_k'$ where $\{j, k\} = \{1, 2\}$ and T_k' is a prefix of T_j. By Theorem A.1, for each W' in T, $\Gamma \vdash W'$, which implies that $\lfloor W' \rfloor_1 \approx_L \lfloor W' \rfloor_2$. Therefore, we have $\Gamma \vdash T_j \approx_L T_k'$. Thus, s satisfies the noninterference property. □