Security Criteria for a Transparent Encryption Layer

(Work in Progress)

Konstantinos Kallas
University of Pennsylvania
Philadelphia, USA

Clara Schneidewind
TU Wien
Vienna, Austria

Benjamin C. Pierce
University of Pennsylvania
Philadelphia, USA

Steve Zdancewic
University of Pennsylvania
Philadelphia, USA

***Abstract*—We study cryptographically-masked flows in the presence of key leakage in an interactive setting. Focusing on the encryption layer in a client-server setting, we propose a range of correctness criteria embodying varying constraints on client and server behavior and assumptions about the power of attackers. We formalize these definitions in Coq and prove that a minimal encrypting middlebox satisfies the strongest property.**

I. Introduction

We want to reason about the security properties of servers that use cryptographic encryption layers (e.g. SSL) to secure communications with clients. A schematic representation of such a server is given in Figure 1. Conceptually, the encryption layer should be decoupled from the server-side logic: its only purpose is encrypting and decrypting incoming and outgoing communications. Conversely, the server logic should know nothing about encryption, key management, etc...

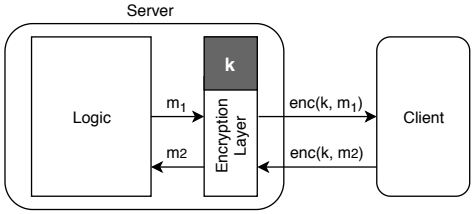

Fig. 1. Schematic illustration of a server with an encryption layer: Incoming messages that represent valid encryptions with shared secret key k are decrypted by the encryption layer and forwarded to the server logic. Outgoing messages produced by the server logic are encrypted with k before being sent to the client.

Formal reasoning about the security of such servers must account for the special nature of cryptographically-masked flows [5]. In particular, in this setting, encryption can ensure message confidentiality only if the cryptographic key is kept secret. If a key is leaked during the interaction, then an attacker can use it to break the confidentiality of messages encrypted with that key.

This paper aims to develop strong, general, and reasonably realistic security properties that intuitively ensure desirable features of servers with encryption layers. In particular, we want to formally characterize the confidentiality of encrypted communications and the absence of (partial) cryptographic key leakage. The challenging part about this context is that it combines cryptographic flows with key leakage in an *interactive* setting, where inputs are provided over time, and can trigger new outputs. This combination raises challenging questions related to the specification of the environment of the server (attacker and client) and (to our knowledge) has not been addressed before in the literature.

More precisely, it turns out that using standard security notions for cryptographically-masked flows in the presence of key leakage without restricting the environment behavior, would deem seemingly secure servers (e.g. a server with an encryption layer that rejects unencrypted messages by sending them back to the client) to be insecure. This opens up the question on which kind of restrictions on environment inputs are realistic, whether they lead to achievable security properties, and how they relate to attacker and client behaviors.

Our contributions in this paper are:

- We define security properties for transparent server-side encryption layers (Section VI). In contrast to the existing work on information flow control, we are (to our knowledge) the first to study cryptographically-masked flows in the presence of key leakage in an *interactive* setting. We also describe a way of expressing attacker capabilities by restricting what the attacker cannot do, instead of specifying what they can do (Section V).
- We construct a minimalistic cryptographic middlebox that encapsulates a server, encrypting its outgoing messages and decrypting incoming messages. We prove this middlebox to satisfy the presented security notions for a class of servers that satisfy a structural non-interference property (Section VII).
- We formally specify the presented security notions and the accompanying infrastructure in the Coq proof assistant. We are in progress of finalizing the mechanized proofs of the results presented in this paper within our framework.

To focus attention on fundamentals, we simplify the problem by omitting many important aspects of such scenarios such as authentication, key exchange, and protocol negotiation. Also, we consider a single-client setting with exactly one secret key that has been pre-shared between the server and an honest client.

We overview background in Section II, present key ideas in Section III, summarize related work in Section VIII, and discuss limitations and possible future work in Section X.

II. Background

The information flow of systems that support cryptographic primitives has been studied under the name *cryptographically-masked flows* [5]. Defining non-interference-based security properties in the presence of cryptographically-masked flows raises special challenges that have been extensively discussed by Askarov *et al.* [5], [6]. In this section we briefly overview these results, before presenting the key ideas of our work in Section III.

A. Non-Interference

Non-interference is a standard notion for describing strong security properties such as confidentiality or integrity for input-output systems or programs. More precisely, non-interference is a hyperproperty [15] that relates pairs of program executions: secrecy is usually expressed using non-interference by requiring that two runs that only differ in secret (or high confidentiality) inputs should not be distinguishable by observing their public (or low confidentiality) outputs. Formally, this notion is typically captured by defining two similarity relations called *low input equivalence* (written $\sim_I$) and *low output equivalence* (written $\sim_O$) on a program configuration Σ. In the case of secrecy, low input equivalence relates all configurations that differ only in their secret (or confidential) values, while low output equivalence relates configurations that are indistinguishable from the perspective of an attacker who can only observe public (low) output values.

B. Occlusion

Defining non-interference in the presence of cryptographic encryption comes with a particular challenge: Low output equivalence should respect that messages m, which are encrypted with a secure encryption scheme using a secret key k (written $\mathit{enc}_{\mathsf{k}}(m)$), are not distinguishable by an attacker. However, as encrypted ciphertexts are just another kind of message, it should be straightforward for an attacker to distinguish whether the very same cipher text is sent twice. This leads to a problem commonly referred to as *occlusion* [5].

Consider the example shown in Figure 2.

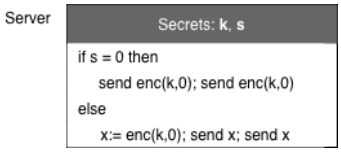

Fig. 2. Simple server with secret key k and secret s that leaks information about s by occlusion. Note that each invocation of enc(k,0) corresponds to the creation of a fresh encryption (with fresh random coins).

An attacker can easily learn some information about the secret s (violating the secrecy of s) by observing the server's output: If the attacker observes that the exact same value is sent two times in sequence, they can estimate that the value of s will not be equal to 0 with overwhelming probability. To account for this issue, Askarov et al. [5] extend encryptions to not only depend on a key k and a message m, but also on an initial vector r (that can be thought of as random coins used in probabilistic encryption schemes).

For initial vectors r one can then characterize the following equality ($\doteq$) on ciphertexts that needs to be respected by the low output equivalence [5]:

$$\forall\ \mathsf{k}_1\ \mathsf{k}_2\ m_1\ m_2\ r_1\ r_2.\quad r_1 = r_2 \iff \mathit{enc}_{\mathsf{k}_1,r_1}(m_1) \doteq \mathit{enc}_{\mathsf{k}_2,r_2}(m_2)$$

This characterization of $\doteq$ ensures that (for same value of r) ciphertexts encrypting different messages can be indistinguishable (for same value of r), but at the same can be distinguished (for different values of r). This allows us to account for the previously shown occlusion problem when using a particular notion of non-interference called *possibilistic non-interference*.

C. Possibilistic Non-Interference

The idea of possibilistic non-interference is that, for every two low-input-equivalent program configurations, the (observable) behavior of a run of the program on one of the configurations can be matched by a run on the other configuration. More formally, considering that $\Sigma \downarrow \Sigma'$ denotes that configuration Σ evaluates to Σ', possibilistic non-interference statements are of the form:

$$\forall\ \Sigma_1\ \Sigma_2.\ \Sigma_1 \sim_I \Sigma_2 \to \Sigma_1 \downarrow \Sigma_1' \to \exists\ \Sigma_2'.\ \Sigma_2 \downarrow \Sigma_2' \to \Sigma_1' \sim_O \Sigma_2'$$

Note that the concrete semantics of non-interference heavily depends on the definitions of Σ, $\sim_I$, and $\sim_O$. For capturing secrecy, low input equivalence should relate such configurations that only differ in secret values For expressing reactive non-interference, configurations should model input and output streams that the low input and output equivalences can be applied to. Qualifying input and output as confidential can e.g., be achieved by an explicit labeling [5].

Applying the idea of possibilistic non-interference to the example in Figure 2, we would require that the set of traces produced by the server for any value of secret s_1 is output equivalent to the set of traces produced by any other secret value s_2. In particular each run of a server with $s = 0$ would need to be matched by a server with $s = 1$. Making the randomness of the encryption explicit, $\mathit{enc}_{\mathsf{k},r_1}(0) \cdot \mathit{enc}_{\mathsf{k},r_2}(0)$ for some $r_1 \neq r_2$ would be a valid trace for the server with $s = 0$. This trace, however cannot be matched by a low output equivalent trace of the server with $s = 1$ as such a server can only produce traces of the form $\mathit{enc}_{\mathsf{k},r}(0) \cdot \mathit{enc}_{\mathsf{k},r}(0)$. Hence, the server would be considered to violate the secrecy of s.

III. Key Ideas

The goal of this work is to study security properties for interactive servers (such as web servers) with a simple transparent encryption layer. We want strong—but reasonable—security notions that apply to a general setting, and to this end we want to limit both the assumptions we make about the server logic and the restrictions we place on the adversarial environment. In particular, we state our definitions in terms of *interaction traces* between the server and its environment, as

opposed to fixing a specific model for representing servers, clients, or attackers. Similarly, we characterize adversarial environments in terms of their limitations, rather than constructively specifying their possible actions; this reduces the risk of under-specifying adversarial capabilities and gives us stronger guarantees.

In this section we overview the key ideas of the paper, starting from the security properties that we are aiming for, followed by a discussion on the issues that arise from the particular setting under consideration (cryptographically-masked flows in the presence of key leakage for interactive servers). We close with a brief discussion on the specification of attacker capabilities.

A. Security Properties

When studying the security properties of a server with an encryption layer, we are mostly interested in the secrecy of data that was exchanged via the encrypted connection (more precisely, those messages that were encrypted with a shared secret key k). From now on, we will refer to this notion of secrecy as *confidential data secrecy*.

Intuitively, confidential data secrecy should be expressible by a single non-interference-based definition; however, it turns out that, in the setting of cryptographically masked flows with key leakage, such a direct characterization is not easy: Since cryptographic keys are modeled symbolically, the notion of key leakage that is commonly used [6] is only concerned with leakage of full keys, not with partial or gradual leakage of key information. Hence, expressing confidential data secrecy as a non-interference property using this model of key leakage would only imply a weak notion of key secrecy (namely that that the key used for encryption might not be leaked *as a whole*, as otherwise an attacker could use this key to decrypt confidential messages). However, a server that leaks its secret key bit by bit would be considered secure by such a definition. We address this issue by explicitly demanding keys that are used for encrypting secure connections to satisfy a strong secrecy propperty.

For this reason the two security properties for web servers with shared secret k that we want to focus on in this work are the following notions of strong secrecy:

1) The secrecy of server-side secret keys (more precisely the shared private key with a client). We will call this property *strong key secrecy*.
2) The secrecy of input values that were sent under encryption with the shared private key. We will call this property *strong confidential data secrecy*.

Strong key secrecy corresponds to the standard notion of strong secrecy [3] applied to keys that are used by a server and its clients to encrypt their communications. Intuitively, strong key secrecy means that the server doesn't leak any information about the key that it uses to encrypt the communication with its client. Correspondingly, strong confidential data secrecy will be captured by a reactive non-interference notion that requires that no information about encrypted inputs sent by the client can be leaked.

B. Restricting Attacker Capabilities

There are several issues that arise when specifying strong secrecy properties for interactive servers in the presence of encryption, and some of them have been separately solved in the setting of cryptographically masked flows without key leakage for interactive systems [5] and the setting of cryptographically masked flows with key leakage for non-interactive systems [6]. However, it turns out that in the combined setting, the previous solutions are not adequate anymore. Below, we try to briefly describe the arising issues, by presenting illustrating examples of problematic interactions.

a) Guessing Secrets: A typical question that is raised when modeling strong secrecy for interactive systems is how to prevent an attacker from guessing secrets. Consider the server with secret key k depicted in Figure 3.

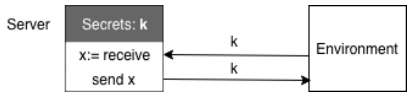

Fig. 3. Simple interactive server with secret key k that sends back any received value.

This server should intuitively be considered secure (in the sense of strong secrecy) as it does not leak any information about the secret key k. However, if the attacker is able to guess the key k, then k will be sent out in plain by the server. This issue can be tackled by treating all values that are not k, to be dissimilar from k in the low input equivalence relation $\sim_I$. Therefore, the set of low input equivalent runs contains a single run, thus trivially satisfying low output equivalence due to reflexivity.

b) Key Leakage: In the presence of encryption, messages that are encrypted with a secret key should be considered low output equivalent as they cannot be distinguished by an attacker. However, an attacker could learn information about a key used for encryption while interacting with the server, and later on use this information to distinguish encrypted messages. Consider the example in Figure 4. This server is

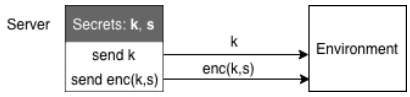

Fig. 4. Simple server with secret key k and secret s that first leaks secret key k and then sends out the encryption of s under k.

clearly insecure as it leaks the secret key k and afterwards sends the ciphertext of some secret s encrypted under k. To account for that, the low output equivalence $\sim_O$ should be sensitive to the set of keys (from now on referred to as *key knowledge*) that an attacker could deduce while interacting with the server. Note that the same issue arises when the key k is leaked after the ciphertext $enc(k, s)$.

Following prior work [6], this can be captured by making the low output equivalence *trace sensitive*. Two ciphertexts using the same key are considered equivalent, only if the key

used to decrypt them cannot be derived from an adversary after observing the whole interaction with the server.

c) Key Leakage by Guessing Secrets: The previously described notion of low output equivalence alone is not sufficient to correctly characterize security when combining the previous two examples as depicted in Figure 5.

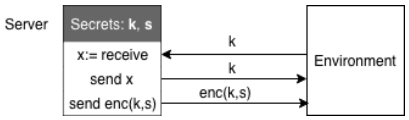

Fig. 5. Simple server with secret key k and secret s that sends back the first message it receives and then sends out the encryption of s under k.

The depicted server holds two secret values: a secret key k and a confidential value s. Intuitively, this server should be secure as it simply sends back the first message that it receives from the environment and afterwards sends the secret encrypted under the secret key k hence not leaking anything about s (or k). However, assuming that an attacker can increase their key knowledge by observing the server output, in the depicted run an attacker that could guess k would learn k from the server output. This would allow them to decrypt the ciphertext that they learns in the next step and thereby to violate the secrecy of s.

This issue cannot be solved by refining the notion of similarity between two arbitrary input traces as done before, but requires to limit the set of possible input traces in the first place. To account for this, we specify the capabilities of an attacker, thus restricting the possible input traces, by describing what a realistic attacker should not be able to do. Intuitively, the attacker should not be able to produce a message that, if observed, would increase their key knowledge. In the depicted example such a property would be trivially violated as sending out key k would imply that k is in the attacker's key knowledge.

C. Attackers

We characterize attackers as the sets of traces that they can produce given an initially known set of cryptographic keys. For example, in order to prevent the issue described in Figure 5, we restrict traces where the attacker sends keys that they don't know in plain text. Our attacker characterizations fall into two separate categories:

1) The interaction of server with an attacker alone.
2) The interaction of server with an attacker in the presence of other (honest) participants concurrently communicating with the server.

While the first scenario focuses on whether an attacker can trick a server into leaking (information about) the secret key that they shares with an honest user, the second scenario additionally models concurrent ongoing sessions with other (honest) clients. Satisfying secrecy properties in the second setting gives particularly strong guarantees, as they imply that no interaction of honest clients and adversaries with the server, could lead to the leakage of encrypted message contents, or of the shared encryption key.

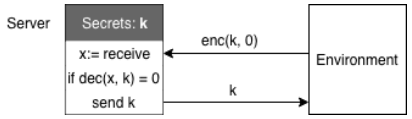

Fig. 6. Simple server with secret key k that leaks key upon receiving a correct encryption of message 0..

In particular, the server depicted in Figure 6 would be considered insecure in this setting for two reasons:

1) An attacker would learn that the value sent by the (honest) client was actually an encryption of message 0 (violating strong confidential data secrecy)
2) An attacker would learn the secret key k (violating strong key secrecy).

All of the above scenarios can be modeled by explicitly tracking the attacker's knowledge of keys and using that knowledge to proscribe the sets of traces that would be impossible to produce by an attacker limited to that knowledge. Below, we formulate such a notion of "knowledge" and use it to define several characterizations of attacker capabilities.

IV. Definitions

We begin by describing our assumptions about the underlying cryptographic model. First of all, we only consider symmetric probabilistic encryption schemes. The encryption algorithm of a probabilistic encryption scheme, is a function $\mathcal{E}$ from a key, a plaintext, and some randomness (referred to as an initial vector). The scheme is called probabilistic, because the algorithm can produce a set of ciphertexts for each pair of key and plaintext, depending on the given initial vector. We assume the decryption algorithm $\mathcal{D}$ is deterministic, and that it fails if a wrong key is used. Formally the decryption function is defined as follows.

$$\forall r, \mathcal{D}(\mathsf{k}, \mathcal{E}(\mathsf{k}', v, r)) = \begin{cases} v, & \text{if } k = k' \\ \bot, & \text{otherwise} \end{cases}$$

In addition, we assume Shannon's perfect secrecy [22], meaning that an observer can learn no information about the plaintext or the key, just by observing its ciphertext.

A. Traces

The notions in this paper are defined in terms of the observable interactions between a server and its environment. The attacker is encapsulated by the environment and can observe the network interactions of the server, that is, the messages that it sends and receives. Messages $m \in \mathcal{M}$ can contain arbitrary values n in plain text, keys k in plain text, ciphertexts $enc_{\mathsf{k},r}(m)$ which represent the result of $\mathcal{E}(\mathsf{k}, v, r)$, or pairs of messages. The formal definition of the message data type $\mathcal{M}$ can be seen below:

$$\mathcal{M} \ni m, m_1, m_2 := n \mid \mathsf{k} \mid enc_{\mathsf{k},r}(m) \mid (m_1, m_2) \quad \mathsf{k}, n \in \mathbb{N}$$

The interaction of the server with the environment is represented as a finite sequence of messages that were sent from, or received by the server. Each message sent or received is represented as an event, which can either be of type *send* or *receive*. Formally:

$$\mathcal{E} \ni e := \mathsf{send}(m) \mid \mathsf{recv}(m) \qquad m \in \mathcal{M}$$

We will call the sequence of events an *event trace* and assume (indexed) variables s to range over event traces.

We introduce the following notions for reasoning about the set representations of messages in an event trace:

$$\begin{aligned} o(s) &:= \{m \mid \exists e.\ e = \mathsf{send}(m)\} && \textit{output messages} \\ i(s) &:= \{m \mid \exists e.\ e = \mathsf{recv}(m)\} && \textit{input messages} \\ \mathit{msg}(s) &:= i(s) \cup o(s) && \textit{trace messages} \end{aligned}$$

Note that the set representation of traces is only used to define the key knowledge that an adversary can acquire from observing a trace, and not for the equivalence based security definitions.

B. Attacker Knowledge and Derivability

In the standard setting, non-interference is defined with respect to a similarity notion, that represents the equivalence of two observed events from the viewpoint of an observer. However, in a setting where events could contain messages encrypted under some key, a static notion of equivalence is not adequate, as the indistinguishability of two events depends on the keys that an observer knows of, and this can change over time. For example an observer who only knows key k cannot distinguish between messages $\mathit{enc}_{\mathsf{k}_1,r_1}(m_1)$ and $\mathit{enc}_{\mathsf{k}_2,r_2}(m_2)$, when $\mathsf{k} \neq \mathsf{k}_1$ and $\mathsf{k} \neq \mathsf{k}_2$, even if $\mathsf{k}_1 = \mathsf{k}_2$. However, an observer who knows both $\mathsf{k}_1, \mathsf{k}_2$ can distinguish the two ciphertexts, by decrypting them and inspecting the encrypted values. Therefore the equivalence notion has to be parametrized by the keys that the attacker knows and can use for decrypting (or encrypting) ciphertexts. We call the set of keys κ that the attacker knows *key knowledge*.

To describe the keys that an attacker can learn from a trace, we define a derivability relation $\cdot, \cdot \vdash \cdot, \cdot$ on pairs of traces and key knowledge sets. We will refer to pairs of the form (tr, κ) as *knowledge states* and call *tr* the *working trace* and κ the *key knowledge*. Note that *tr* here denotes the set representation of a trace. This is adequate as the order of events does not affect the key knowledge that can be derived from a trace, as the observer has recorded the whole trace, and can read it in any order.

$$\frac{}{\mathit{tr}^i, \kappa^i \vdash \mathit{tr}^i, \kappa^i} \qquad \frac{\mathsf{k} \in \mathit{tr} \quad \mathit{tr}^i, \kappa^i \vdash \mathit{tr}, \kappa}{\mathit{tr}^i, \kappa^i \vdash \mathit{tr}/\{\mathsf{k}\}, \{\mathsf{k}\} \cup \kappa}$$

$$\frac{\mathit{enc}_{\mathsf{k},r}(m) \in \mathit{tr} \quad \mathsf{k} \in \kappa \quad \mathit{tr}^i, \kappa^i \vdash \mathit{tr}, \kappa}{\mathit{tr}^i, \kappa^i \vdash \{d\} \cup \mathit{tr}/\{\mathit{enc}_{\mathsf{k},r}(d)\}, \kappa}$$

$$\frac{(m_1, m_2) \in \mathit{tr} \quad \mathit{tr}^i, \kappa^i \vdash \mathit{tr}, \kappa}{\mathit{tr}^i, \kappa^i \vdash \{m_1, m_2\} \cup \mathit{tr}/\{(m_1, m_2)\}, \kappa}$$

Intuitively, $\mathit{tr}^i, \mathit{tr} \vdash \kappa^i, \kappa$ holds when given initial key knowledge κ^i, an observer can deconstruct the trace tr^i to trace *tr* and in the process learn new keys, thus ending up with key knowledge κ. The applications of the presented rules can be seen as the steps that an attacker with knowledge κ^i would perform to extract all extractable keys from trace tr^i. In this process, tr^i is transformed to *tr* by deconstructing composed messages (pairs), decrypting encrypted data in case that the encryption key is known, and adding plain keys to the accumulated key knowledge.

The presented derivability notion is monotone in the key knowledge: κ^i is a subset of κ, as the knowledge can only increase when observing a trace. In contrast, traces are drained: intuitively *tr* contains less "extractable" information than tr^i. We make these intuitions precise, by formulating derivability as a rewrite system on knowledge states.

$$\begin{aligned} (\{\mathsf{k}\} \cup \mathit{tr}, \kappa) &\to (\mathit{tr}/\{\mathsf{k}\}, \{\mathsf{k}\} \cup \kappa) \\ (\{\mathit{enc}_{\mathsf{k},}(m) \cup \mathit{tr}, \{\mathsf{k}\} \cup \kappa) &\to (\{m\} \cup \mathit{tr}/\{\mathit{enc}_{\mathsf{k},}(n), \{\mathsf{k}\} \cup \kappa) \\ (\{(m_1, m_2)\} \cup \mathit{tr}, \kappa) &\to (\{m_1, m_2\} \cup \mathit{tr}/\{(m_1, m_2)\}, \kappa) \end{aligned}$$

Note that each rewrite rule removes an element from the initial trace. We can use those elements to characterize the concrete *action* taken in each rewrite step. We optionally annotate the rewrite relation with this action.

Extending standard notions for abstract rewrite systems, we say that a knowledge state (tr, κ) is *unreducible* if there is no (tr', κ') such that $(\mathit{tr}, \kappa) \to (\mathit{tr}', \kappa')$. Accordingly we will say that $(\mathit{tr}^i, \kappa^i)$ reduces to normal form (tr, κ) (written $(\mathit{tr}^i, \kappa^i) \downarrow_{\mathit{max}} (\mathit{tr}, \kappa)$), if $(\mathit{tr}^i, \kappa^i) \to^* (\mathit{tr}, \kappa)$ and (tr, κ) is unreducible where $\to^*$ denotes the reflexive and transitive closure of $\to$.

In order to establish a confluence result, we have to formally capture the possible deviation of a knowledge state after applying two different steps. It turns out that two different rewriting steps cannot be easily joined within one step as illustrated in the following example:

Example 1:

$$(\{\mathsf{k}_1, (\mathsf{k}_1, \mathsf{k}_2)\}, \emptyset) \xrightarrow{(k_1,k_2)} (\{\mathsf{k}_1, \mathsf{k}_2\}, \emptyset) \xrightarrow{k_1} (\{\mathsf{k}_2\}, \{\mathsf{k}_1\})$$

$$(\{\mathsf{k}_1, (\mathsf{k}_1, \mathsf{k}_2)\}, \emptyset) \xrightarrow{k_1} (\{(\mathsf{k}_1, \mathsf{k}_2)\}, \{\mathsf{k}_1\}) \xrightarrow{(k_1,k_2)} (\{\mathsf{k}_1, \mathsf{k}_2\}, \{\mathsf{k}_1\})$$

More precisely, it is possible to join the key knowledge within one step, but not necessarily the working trace. However, we can capture the gap between the sets by a looser equivalence relation that we call *mutual deconstruction*.

Intuitively, a valid *deconstruction* of a message m with respect to a key knowledge κ, is a decomposition that preserves all atomic components of m upto keys in κ. Formally we describe deconstructions by the following predicate:

$$\frac{}{m \rightsquigarrow^\kappa \{m\}} \qquad \frac{\mathsf{k} \in \kappa}{\mathsf{k} \rightsquigarrow^\kappa \emptyset} \qquad \frac{\mathsf{k} \in \kappa \qquad m \rightsquigarrow^\kappa \mathit{tr}}{\mathit{enc}_{\mathsf{k},r}(m) \rightsquigarrow^\kappa \mathit{tr}}$$

$$\frac{m_1 \rightsquigarrow^\kappa \mathit{tr}_1 \qquad m_2 \rightsquigarrow^\kappa \mathit{tr}_2}{(m_1, m_2) \rightsquigarrow^\kappa \mathit{tr}_1 \cup \mathit{tr}_2}$$

By lifting deconstruction to traces, we can establish a new equivalence relation on trace knowledge states.

Definition 1 (Trace Deconstruction): Trace tr_1 deconstructs trace tr_2 under knowledge κ (written $tr_1 \rightsquigarrow^{\kappa} tr_2$) iff

$$\forall m.\ m \in tr_1 \rightarrow \exists tr.\ m \rightsquigarrow^{\kappa} tr\ \wedge\ tr \subseteq tr_2.$$

Definition 2 (Mutual Trace Deconstruction): Traces tr_1 and tr_2 mutually deconstruct each other under knowledge κ (written $tr_1 \leftrightsquigarrow^{\kappa} tr_2$) iff

$$tr_1 \rightsquigarrow^{\kappa} tr_2\ \wedge\ tr_2 \rightsquigarrow^{\kappa} tr_1$$

With respect to mutual trace deconstruction we can prove confluence of the given abstract rewrite system.

Lemma 1 (Confluence): Let $(tr, \kappa) \rightarrow^* (tr_1, \kappa_1)$ and $(tr, \kappa) \rightarrow^* (tr_2, \kappa_2)$. Then there exist κ^*, tr_1^*, and tr_2^* such that $(tr_1, \kappa_1) \rightarrow^* (tr_1^*, \kappa^*)$, $(tr_2, \kappa_2) \rightarrow^* (tr_2^*, \kappa^*)$ and $tr_1^* \leftrightsquigarrow^{\kappa^*} tr_2^*$.

From this result we can prove the existence of unique normal forms:

Lemma 2 (Existence of unique normal forms): For every (tr, κ) there is a unique normal form (tr_{max}, κ_{max})

V. Attacker Capabilities

In contrast to classical reactive non-interference notions, which typically assume unrestricted attackers [14], the setting of cryptographically-masked flows requires a restriction of an interactive attacker's capabilities. An unrestricted attacker would correspond to a computationally unbounded one and hence could easily break any cryptographic primitive.

Instead of defining the attacker capabilities in a constructive way, that is by specifying what the attacker can do (construct and send messages to the server, destruct observed messages from the server, etc.), we would like to specify attacker capabilities in a non-constructive way. The rationale behind this decision is that a non-constructive attacker specification approaches the attacker from an overapproximation of its behaviour, therefore reducing the risk of underspecifying attacker capabilities. At the same time we would prefer to state attacker capabilities independently from the concrete attacker representation. More precisely, we don't want the attacker model to be tied to a specific language or process calculus, but to be purely defined on interaction traces with the server.

In this paper we propose several different trace-based specifications of attacker capabilities with respect to the attacker's key knowledge (or knowledge state). All attacker notions that we present characterize the potential actions of an adversarial environment that encompasses *active attackers* who interact with the system in question. Note that when referring to attacker capabilities in the remainder of the paper we comprise the full environment's actions (including the clients' actions). Taking this unified view allows for simple, uniform specifications of potential inputs to the server and hence simplifies reasoning about the server's security.

The presented notions differ in the amount of knowledge that the attacker may use for for creating their messages to the system. Accordingly, we formulate attacker capabilities as predicates on traces parametrized by the attacker's initial key knowledge κ^i.

A. Attacker Capabilities

We specify attacker capabilities along two dimensions: Their ability of sending messages that they constructed themselves and their ability of sending messages that were constructed by other entities. We refer to the former as *attacker-constructed messages* and to the latter as *client constructed messages.* Both attacker-constructed as well as client constructed messages are defined with respect to a key knowledge. Intuitively, those ciphertexts that are encrypted with keys known to the attacker are considered attacker-constructed, while ciphertexts that were encrypted with unknown keys are considered client constructed.

1) Attacker-constructed Messages: The general idea of restricting attacker-constructed messages is to ensure that the attacker did not use keys that they didn't know when constructing some message. This is important, as it establishes that an attacker can't simply guess keys and hence provoke situations as described in Section III-B. For expressing this, we can use the intuition that the maximal key knowledge κ_{max} of trace tr^i with initial knowledge κ^i actually gives the set of keys that were used in attacker-constructed messages. Ensuring that κ_{max} does not contain any previously unknown message therefore effectively restricts attacker traces not to use inaccessible knowledge.

By directly applying this intuition, we arrive at the following (non-adaptive) definition of possible attacker traces with attacker-constructed messages restricted by knowledge κ^i.

$$\mathcal{T}^{attacker}_{non\text{-}adapt}(\kappa^i) := \{s \mid (i(s), \kappa^i) \downarrow_{max} (tr_{max}, \kappa_{max})\ \wedge\ \kappa_{max} \subseteq \kappa^i\}$$

Note that this definition is non-adaptive in the sense that it considers the initial knowledge κ^i as static reference along the trace 1) for qualifying (sub) messages as attacker-constructed and 2) as reference for knowledge growth.

An adaptive definition takes into account that the attacker's key knowledge might grow by observing messages that were sent by the server. We capture this by requiring that every message m sent by the attacker (hence is a receive event) should not increase the attacker's key knowledge (as compared to the key knowledge before the receive event). This intuition is formalized by the following definition:

$$\begin{aligned}\mathcal{T}^{attacker}_{adapt}(\kappa^i) := \{s \mid \forall s_{pre}\ s_{post}\ m.\ s = s_{pre} \cdot [\mathsf{recv}(m)] \cdot s_{post} \rightarrow \\ (msg(s_{pre}), \kappa^i) \downarrow_{max} (tr^{pre}_{max}, \kappa^{pre}_{max}) \rightarrow \\ (\{m\}, \kappa^{pre}_{max}) \downarrow_{max} (tr_{max}, \kappa_{max}) \rightarrow \kappa_{max} \subseteq \kappa^{pre}_{max}\}\end{aligned}$$

This definition can be seen as a point-wise version of the non-adaptive definition where each receive event is restricted by the knowledge κ^{pre}_{max} gained from the prefix trace instead of the initial knowledge κ^i.

Note that the key knowledge in this setting is inter-dependent: 1) knowledge is used for classifying messages into attacker-constructed and client constructed ones (because whether the content of an encrypted message is considered for

knowledge generation depends on whether the encryption key is contained in the attacker's key knowledge) 2) knowledge evolves according to the (decryptable) trace messages and a growth in key knowledge again changes the messages considered for key generation.

For these reason the two presented notions are also incomparable. Consider the example in Figure 7.

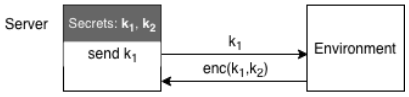

Fig. 7. Simple server with secret keys k_1 and k_2 that leaks k_1.

This server leaks the key k_1. Assuming an adaptive attacker that initially doesn't know k_1 or k_2, the message $enc_{\mathsf{k}_1,r}(\mathsf{k}_2)$ would not be allowed to be sent after receiving k_1 as doing so would increase the attacker's knowledge given that they already learned k_1 from the prior interaction. Statically assuming the empty key knowledge sending $enc_{\mathsf{k}_1,r}(\mathsf{k}_2)$ would not increase the key knowledge as $(\{enc_{\mathsf{k}_1,r}(\mathsf{k}_2)\}, \emptyset) \downarrow_{max} (\{enc_{\mathsf{k}_1,r}(\mathsf{k}_2)\}, \emptyset)$.

On the other hand, there are clearly inputs forbidden by a non-adaptive attacker that are allowed by a non-adaptive definition as shown by the interaction depicted in Figure 8. Assuming the knowledge from the previous interaction (so

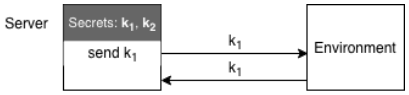

Fig. 8. Simple server with secret keys k_1 and k_2 that leaks k_1.

the server message k_1), an attacker sending key k_1 would not increase it's key knowledge. However, assuming an initial empty knowledge sending k_1 would cause an increase in the key knowledge, hence be forbidden by $\mathcal{T}^{attacker}_{non\text{-}adapt}(\emptyset)$.

This issue gives rise to a more liberal definition of attacker capabilities that performs message classification according to the initial knowledge, but judges the key growth with respect to the adaptive key knowledge:

$$\begin{aligned}\mathcal{T}^{attacker}_{semi\text{-}adapt}(\kappa^i) := \\ \{s \mid \forall s_{pre}\ s_{post}\ m.\ s = s_{pre} \cdot [\mathsf{recv}(m)] \cdot s_{post} \rightarrow \\ (msg(s_{pre}), \kappa^i) \downarrow_{max} (tr^{pre}_{max}, \kappa^{pre}_{max}) \rightarrow \\ (\{m\}, \kappa^i) \downarrow_{max} (tr_{max}, \kappa_{max}) \rightarrow \kappa_{max} \subseteq \kappa^{pre}_{max}\}\end{aligned}$$

This definition would allow for the both interactions depicted in Figure 7 and Figure 8. This is as $enc_{\mathsf{k}_1,r}(\mathsf{k}_2)$ would be assumed to be client constructed with respect to the initial empty key knowledge in Figure 7 and hence would not subject to restriction. On the other side, in Figure 8, sending out k_1 would not cause an increase in knowledge with respect to the knowledge learned by the previous interaction (which would be exactly $\{\mathsf{k}_1\}$).

2) Client Constructed Messages: While attacker-constructed messages are restricted by the maximal key knowledge that they expose, client constructed messages show up as encryptions in the maximal working trace of the environment inputs and can be restricted by putting limitations on that maximal working trace.

For restricting client constructed messages we need to consider two aspects: 1) which key knowledge is considered for classifying messages as client constructed. As done for attacker-contructed messages we need to make a distinction into adaptive and a non-adaptive knowledge used for classification. 2) whether one should allow for replaying messages that were already observed, so the kind of restriction put on client constructed messages.

For simplifying the characterization of client constructed messages, we will let in the following denote $\text{ENCS}(tr)$ the encryptions in trace *tr*. More formally:

$$\text{ENCS}(tr) := \{m_{enc} \in tr_{max} \mid \exists k\ r\ m.\ m_{enc} = enc_{\mathsf{k},r}(m)\}$$

First we consider a non-adaptive notion where (non-decryptable) encryptions are excluded:

$$\begin{aligned}\mathcal{T}^{client}_{non\text{-}adapt}(\kappa^i) := \\ \{s \mid (i(s), \kappa^i) \downarrow_{max} (\kappa_{max}, tr_{max}) \ \wedge\ \text{ENCS}(tr_{max}) = \emptyset\}\end{aligned}$$

Note that the classification of messages to be be client constructed is based on the initial knowledge κ^i. This means that even learning a key during interaction does not allow for sending encryptions using this key. Considering the interaction with the key-leaking server depicted in Figure 9, sending $enc_{\mathsf{k},r}(m)$ would be forbidden even though the attacker would have sufficient knowledge for creating this encryption himself as she learned k from the interaction. However according to an empty initial knowledge $enc_{\mathsf{k},r}(m)$ will be classified as client constructed and hence be constraint.

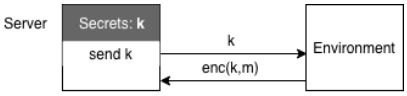

Fig. 9. Simple server with secret keys k that leaks k.

A more realistic notion considers an adaptive key knowledge similar to the one in the specification of $\mathcal{T}^{attacker}_{adapt}(\cdot)$. This notion ensures that once a key is learned an encryption with this key can be created.

$$\begin{aligned}\mathcal{T}^{client}_{adapt}(\kappa^i) := \\ \{s \mid \forall s_{pre}\ s_{post}\ m.\ s = s_{pre} \cdot [\mathsf{recv}(m)] \cdot s_{post} \rightarrow \\ (msg(s_{pre}), \kappa^i) \downarrow_{max} (tr^{pre}_{max}, \kappa^{pre}_{max}) \rightarrow \\ (\{m\}, \kappa^{pre}_{max}) \downarrow_{max} (tr_{max}, \kappa_{max}) \rightarrow \text{ENCS}(tr_{max}) = \emptyset\}\end{aligned}$$

This notion accounts for the interaction shown in Figure 9 as it accounts for the growth in key knowledge by observing the interaction for classifying as $enc_{\mathsf{k},r}(m)$ as attacker-constructed and hence does not constrain it.

TABLE I
OVERVIEW ON POSSIBLE ATTACKERS EXPRESSED AS INTERSECTIONS ON CAPABILITIES FOR RESTRICTING ATTACKER-CONSTRUCTED AND CLIENT CONSTRUCTED MESSAGES.

$\cap$	$\mathcal{T}^{attacker}_{non\text{-}adapt}(\cdot)$	$\mathcal{T}^{attacker}_{adapt}(\cdot)$	$\mathcal{T}^{attacker}_{semi\text{-}adapt}(\cdot)$
$\mathcal{T}^{client}_{non\text{-}adapt}(\cdot)$	$\mathcal{A}_{non\text{-}adapt}(\cdot)$	×	×
$\mathcal{T}^{client}_{adapt}(\cdot)$	×	$\mathcal{A}_{adapt}(\cdot)$	$\mathcal{A}_{semi\text{-}adapt}(\cdot)$
$\mathcal{T}^{client}_{adapt,\ replay}(\cdot)$	×	$\mathcal{A}_{adapt,\ replay}(\cdot)$	$\mathcal{A}_{semi\text{-}adapt,\ replay}(\cdot)$
$\mathcal{T}_{\mathcal{E}}$	$\mathcal{A}_{non\text{-}adapt,\ preplay}(\cdot)$	$\mathcal{A}_{adapt,\ preplay}(\cdot)$	$\mathcal{A}_{semi\text{-}adapt,\ preplay}(\cdot)$

Finally, for allowing an attacker to replay messages that were already seen, we extend the adaptive definition to allow for encryptions in the maximal working trace in case that they were already present in the previous (server) trace.

$$\begin{aligned}\mathcal{T}^{client}_{adapt,\ replay}(\kappa^i) := \\ \{s \mid \forall s_{pre}\ s_{post}\ m.\ s = s_{pre} \cdot [\mathsf{recv}(m)] \cdot s_{post} \rightarrow \\ (msg(s_{pre}), \kappa^i) \downarrow_{max} (tr^{pre}_{max}, \kappa^{pre}_{max}) \rightarrow \\ (\{m\}, \kappa^{pre}_{max}) \downarrow_{max} (tr_{max}, \kappa_{max}) \rightarrow \\ \text{ENCS}(tr_{max}) \subseteq \text{ENCS}(tr^{pre}_{max})\}\end{aligned}$$

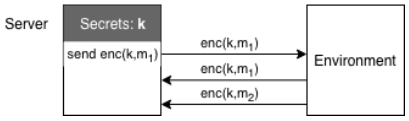

Fig. 10. Simple server with secret keys k sends out $enc_{\mathsf{k},r}(m_1)$.

Considering the interaction depicted in Figure 10, the first attacker message ($enc_{\mathsf{k},r}(m_1)$) will be allowed as it does not increase the knowledge about client constructed messages that was assembled from the previous interaction (which is exactly $\{enc_{\mathsf{k},r}(m_1)\}$). In contrast, the second attacker message ($enc_{\mathsf{k},r}(m_2)$) will be rejected as it adds a new client constructed message.

B. Attackers

Section V-B overviews the different ways of combining the attacker capabilities for attacker-constructed and client-constructed messages. Combinations that are infeasible (due to incompatible levels of adaptability) are marked with ×.

Additionally, in Figure 11 we give a hierarchy of the attackers presented in Table V-B. In this paper we want to focus on two interesting combinations: The semi-adaptive replaying attacker (as a standard attacker from literature) and the semi-adaptive preplaying attacker (as a particularly strong, but still reasonable attacker in the web setting).

1) Semi-adaptive Replaying attacker: A realistic attacker should – in addition to creating arbitrary messages that don't require any knowledge of secret keys – be able to replay encryptions in the case that she has already encountered them during her interaction with the server. This notion is captured by the semi-adaptive replaying attacker, formally defined as follows:

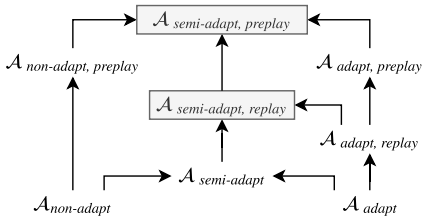

Fig. 11. Hierarchy of different attacker capability characterizations: A connection $\mathcal{A}_1 \rightarrow \mathcal{A}_2$ denotes that $\forall \kappa^i.\ \mathcal{A}_1(\kappa^i) \subseteq \mathcal{A}_2(\kappa^i)$. Attackers that are not transitively connected are considered incomparable.

Definition 3 (Semi-adaptive replaying attacker):

$$\mathcal{A}_{semi\text{-}adapt,\ replay}(s) := \mathcal{T}^{attacker}_{semi\text{-}adapt}(s) \cap \mathcal{T}^{client}_{adapt,\ replay}(s).$$

Intuitively, $\mathcal{T}^{attacker}_{semi\text{-}adapt}(s)$ ensures that messages sent by the attacker may not contain any secret keys (unless they occur encrypted under some other unknown key). The semi-adaptivity ensures that encrypted messages whose keys are learned during the interaction are not considered to contribute to the attacker's key knowledge. Instead the sending of encrypted messages is purely restricted by $\mathcal{T}^{client}_{adapt,\ replay}(s)$ which ensures that those encryptions with initially unknown keys might only be those already learned in the prior interaction with the server.

2) Semi-adaptive Preplaying Attacker: For defining a semi-adaptive preplaying attacker, we don't put any restrictions on client constructed messages, but only prevent attacker-constructed messages to contain guessed keys.

Definition 4 (Semi-adaptive preplaying attacker):

$$\mathcal{A}_{semi\text{-}adapt,\ preplay}(s) := \mathcal{T}^{attacker}_{semi\text{-}adapt}(s)$$

Even though such a preplaying attacker who allows for arbitrary client constructed messages might seem unrealistically powerful on the first sight, it turns out that such a notion actually describes the potential encrypted interactions of an honest user with the server that might be observed by an attacker. For characterizing the security of an encrypted connection between and honest user and server the observation from these kinds of interactions should not allow an attacker to draw any conclusions on the honest user's encrypted messages to the server. Hence modeling arbitrary messages encrypted under keys that are unknown to the attacker is essential for defining a meaningful notion of the (strong) secrecy of confidential data.

On the other hand, it might also be the case, that only the user that established the secure connection could trigger a flawed server into leaking (information about) the encryption key used during this secure connection as depicted in Figure 6.

From this perspective, a preplaying attacker might be considered to control the honest user, thus being able to trigger all actions that an honest user (knowing the key) could perform. This would correspond to an attacker that can trick a user into performing arbitrary actions (as e.g., assumed in CSRF attacks where an attacker makes a user execute unwanted actions).

So intuitively, there are two possible interpretations of this attacker characterization; one where the attacker is able to trick any user into performing arbitrary actions and one where the attacker interacts with the server in an environment that contains honest users that arbitrarily interact with the server. In the first interpretation it is implicitly assumed that the users cannot be tricked to leak any secret keys in plain, whereas in the second interpretation a well-behavedness condition is implicitly assumed from the honest users.

VI. Non-Interference Notions for Servers

Non-interference is a standard security notion for capturing the absence of information leakage in systems. In this work we are concerned with the leakage of information that might corrupt a secure connection between a server and a client. Specifically we are interested in the (strong) secrecy of 1) shared secrets (in particular private keys) between the server and the client and 2) information sent via the secure connection (messages encrypted with a shared secret key).

A. Strong Key Secrecy

The first notion that we will consider is *strong key secrecy*. Intuitively, strong secrecy means that no information (also no partial information) about a secret might be leaked. This includes leakages caused by implicit flows.

We will state the different security notions for servers in terms of generic templates for possibilistic non-interference on partial traces. The use of partial traces and a possibilistic non-interference definition allows us to naturally account for the termination and occlusion behavior of the server as discussed in Section II-C.

We first give a template for strong key secrecy. We will parametrize the non-interference template with a generic set of servers $\mathcal{X}$, and a corresponding (partial) trace predicate $\mathit{isTrace} \in \mathcal{X} \to \mathcal{P}(\mathcal{T}_\mathcal{E} \times \mathbb{B})$. For every server the trace predicate characterizes the set of the server's traces where a trace also incorporates a boolean flag for indicating whether the trace is terminating or not.

Definition 5 (Possibilistic Strong Key Secrecy on Partial Traces): Let $\mathcal{X}$ be a set of servers, $\mathit{server} \in \mathcal{K} \to \mathcal{X}$ be a familiy of servers parametrized by a secret, and $\mathit{isTrace} \in \mathcal{X} \to \mathcal{P}(\mathcal{T}_\mathcal{E} \times \mathbb{B})$ be a family of trace predicates over servers in $\mathcal{X}$. Then we define *server* to satisfy strong key secrecy for attacker capabilities $\mathcal{A}$ and low equivalence relation $\overset{L_{out}}{\sim}$ (written: $\mathrm{KSec}^{\mathit{server},\mathit{isTrace}}_{\mathcal{A},\overset{L_{out}}{\sim}}$) as follows:

$$\begin{aligned}\mathrm{KSec}^{\mathit{server},\mathit{isTrace}}_{\mathcal{A},\overset{L_{out}}{\sim}} := \forall \kappa_1^i\ \kappa_2^i\ \mathsf{k}_1\ \mathsf{k}_2\ s_1\ r_1.\\ (s_1, r_1) \in \mathit{isTrace}(\mathit{server}(\mathsf{k}_1))\\ \to \kappa_1^i = \mathcal{K}/\{\mathsf{k}_1\} \to \kappa_2^i = \mathcal{K}/\{\mathsf{k}_2\} \to s_1 \in \mathcal{A}(\kappa_1^i)\\ \to \exists\ s_2\ r_2.\ (s_2, r_2) \in \mathit{isTrace}(\mathit{server}(\mathsf{k}_2))\ \wedge\ s_2 \in \mathcal{A}(\kappa_2^i)\\ \wedge\ s_1\ {}_{\kappa_1^i}\overset{L_{out}}{\sim}_{\kappa_2^i}\ s_2\ \wedge\ r_1 = r_2\end{aligned}$$

Intuitively, this non-interference notion describes that an attacker should not leak anything about it's secret. To achieve this, we choose the strong attacker scenario in which the attacker initially knows all keys but the secret one. Note that by requiring equality on the termination indicators r_1 and r_2, we fix the non-interference notion to be *termination-sensitive*: whenever a program run of a server (with a specific secret) can terminate then there must be (a low output equivalent) run for any other secret that also terminates. For generalizing the non-interference template to also account for non-termination-sensitive definitions one could parametrize the template by a generic similarity notions on the termination indicators.

Relevant attackers for Strong Key Secrecy: Strong secrecy is an interesting notion in the presence of non-preplaying attackers: In this context it means that an attacker that is in one (or multiple) sessions with the server cannot trick the server into leaking the honest users key. Still, there are possible scenarios where a key or information about a key might be leaked, but that such an event can only be triggered by a user knowing the secret key. Reconsidering the example from Figure 6 one can see that in this case only a message encrypted by the secret key could trigger the key leakage. Detecting such behaviour would require a preplaying attacker that models honest client actions.

B. Strong Confidential Data Secrecy

Protecting the keys that secure a connection is a necessary, but not sufficient condition of achieving a secure communication between the server and an honest client. Additionally, we would like to require from a secure server also not to leak any contents submitted by a user over a secure connection. We can capture this idea by a reactive non-interference notion that requires that the client's protected inputs can not influence the servers outputs:

Definition 6 (Possibilistic Reactive Non-Interference on Partial Traces): Let $\mathcal{X}$ be a set of servers, $\mathit{server} \in \mathcal{X}$ be a server, and $\mathit{isTrace} \in \mathcal{X} \to \mathcal{P}(\mathcal{T}_\mathcal{E} \times \mathbb{B})$ be a family of trace predicates over servers in $\mathcal{X}$. Then we define *server* to be possibilistic reactive non-interferent with respect to a secret key k, notions of low-input equivalence $\overset{L_{in}}{\sim}$ and low-output equivalence $\overset{L_{out}}{\sim}$ (written $\mathrm{CSec}^{\mathit{server},\mathsf{k},\mathit{isTrace}}_{\mathcal{A},\overset{L_{out}}{\sim},\overset{L_{in}}{\sim}}$) as follows.

$$\begin{aligned}\mathrm{CSec}^{\mathit{server},\mathsf{k},\mathit{isTrace}}_{\mathcal{A},\overset{L_{out}}{\sim},\overset{L_{in}}{\sim}} := \forall \kappa^i\ \mathsf{k}\ s_1\ \mathit{itr}_1\ \mathit{itr}_2\ \mathit{otr}_1.\ \kappa^i = \mathcal{K}/\{\mathsf{k}\}\\ \to (s_1, r_1) \in \mathit{isTrace}(\mathit{server}) \to s_1 \in \mathcal{A}(\kappa^i)\\ \to s_1 = (\mathit{itr}_1 \parallel \mathit{otr}_1) \to \mathit{itr}_1\ {}_{\kappa^i}\overset{L_{in}}{\sim}_{\kappa^i}\ \mathit{itr}_2\\ \to \exists \mathit{otr}_2\ s_2\ r_2.\ s_2 = (\mathit{itr}_2 \parallel \mathit{otr}_2)\\ \wedge\ (s_2, r_2) \in \mathit{isTrace}(\mathit{server})\ \wedge\ s_2 \in \mathcal{A}(\kappa^i)\\ \wedge\ s_1\ {}_{\kappa^i}\overset{L_{out}}{\sim}_{\kappa^i}\ s_2\ \wedge\ r_1 = r_2\end{aligned}$$

where $(\mathit{itr}_1 \parallel \mathit{otr}_1)$ denotes an interleaving of input and output traces (including that input messages are lifted to $\mathsf{send}()$ events and output messages are lifted to $\mathsf{recv}()$ events)

Intuitively, we will consider such inputs low input equivalent that cannot be distinguished by an attacker with initial key knowledge. This will give us that every encrypted connection that was initially protected by a secret key, will not leak

$$\frac{}{m \;{}_{\kappa_1}\overset{\mathcal{M}}{\sim}_{\kappa_2}\; m} \qquad \frac{m_1 \;{}_{\kappa_1}\overset{\mathcal{M}}{\sim}_{\kappa_2}\; m_2 \qquad m_1' \;{}_{\kappa_1}\overset{\mathcal{M}}{\sim}_{\kappa_2}\; m_2'}{(m_1, m_1') \;{}_{\kappa_1}\overset{\mathcal{M}}{\sim}_{\kappa_2}\; (m_2, m_2')}$$

$$\frac{\mathsf{k} \in \kappa_1^i \qquad \mathsf{k} \in \kappa_2^i \qquad m_1 \;{}_{\kappa_1}\overset{\mathcal{M}}{\sim}_{\kappa_2}\; m_2}{\mathit{enc}_{\mathsf{k},r_1}(m_1) \;{}_{\kappa_1}\overset{\mathcal{M}}{\sim}_{\kappa_2}\; \mathit{enc}_{\mathsf{k},r_2}(m_2)}$$

$$\frac{\mathsf{k}_1 \notin \kappa_1^i \qquad \mathsf{k}_2 \notin \kappa_2^i}{\mathit{enc}_{\mathsf{k}_1,r}(m_1) \;{}_{\kappa_1}\overset{\mathcal{M}}{\sim}_{\kappa_2}\; \mathit{enc}_{\mathsf{k}_2,r}(m_2)}$$

$$\frac{}{\epsilon \;{}_{\kappa_1}\overset{L_{in}}{\sim}_{\kappa_2}\; \epsilon} \qquad \frac{m_1 \;{}_{\kappa_1}\overset{\mathcal{M}}{\sim}_{\kappa_2}\; m_2 \qquad s_1 \;{}_{\kappa_1}\overset{L_{in}}{\sim}_{\kappa_2}\; s_2 \qquad \mathsf{e} \in \{\mathsf{send}, \mathsf{recv}\}}{\mathsf{e}(m_1) \cdot s_1 \;{}_{\kappa_1}\overset{L_{in}}{\sim}_{\kappa_2}\; \mathsf{e}(m_2) \cdot s_2}$$

$$\frac{\begin{array}{c}(\mathit{msg}(s_1), \kappa_1) \downarrow_{\mathit{max}} (\mathit{tr}_1, \kappa_{\mathit{max}}^1) \\ (\mathit{msg}(s_2), \kappa_2) \downarrow_{\mathit{max}} (\mathit{tr}_2, \kappa_{\mathit{max}}^2) \qquad s_1 \;{}_{\kappa_{\mathit{max}}^1}\overset{L_{in}}{\sim}_{\kappa_{\mathit{max}}^2}\; s_2\end{array}}{s_1 \;{}_{\kappa_1}\overset{L_{out}}{\sim}_{\kappa_2}\; s_2}$$

Fig. 12. Low input and output equivalence

any information the messages sent via this connection (aka protected with the corresponding secret key).

Relevant attackers for Strong Confidential Data Secrecy: Interactive non-interference only yields a meaningful security notion in the presence of preplaying attackers. Otherwise the messages exchanged between honest (protected) users and the server would not be modeled in the first place.

C. Low equivalences

The previously presented notions only get a concrete meaning when giving instantiations for the notions of low input and low output equivalence. Even though it is possible to give several useful instantiations (e.g. to account for weaker security definitions that allow for leakage due to response patterns) we want to give here some strong intuitive definitions for low input and low output equivalence that will be used in the remainder of the paper. These notions are formally defined by the inference rules given in Figure 12.

Intuitively, we will consider two input traces low input equivalent (written $\overset{L_{in}}{\sim}$) if an attacker with some initial key knowledge κ^i cannot distinguish them. In contrast, we will consider two interaction traces low output equivalent (written $\overset{L_{out}}{\sim}$) if an attacker even when deriving all knowledge that she could learn from the interaction could not distinguish these traces. As the (initial) key knowledge κ^i (which also restricts the input traces according to the attacker definition) might differ in the two runs considered by the non-interference based definitions, low equivalences need to be parametrized by two key knowledge sets: one representing the attacker knowledge in the first run and one representing the knowledge in the second run.

Formally, for the low input equivalence notion will just require pairwise indistinguishability of the messages in the two considered traces. Correspondingly *message indistinguishability* (written $\overset{\mathcal{M}}{\sim}$) is defined as in Figure 12.

Intuitively two encryptions of messages are considered indistinguishable with respect to key knowledge sets κ_1 and κ_2 if either their keys are known (and equal) and the encrypted messages again are indistinguishable or if the keys are both not known (with respect to the corresponding knowledge set) and the same initial vector r was used for creating them. This treatment of indistiguishability for encrypted messages corresponds to the approach taken in [6] and [5]. Making two encryptions only indistinguishable if they share the same initial vector allows for easily finding an indistinguishable encryption for each independent encryption while at the same time accounting for dependencies between different encryptions. We refer the reader to the discussion on occlusion in Section II-B.

Low input equivalence is defined by lifting message equivalence to event traces in a standard way.

Finally, low output equivalence is defined in terms of low input equivalence by replacing the (initial) key knowledge by the maximal derivable knowledge. Two event traces are considered output equivalent if all its messages are pairwise equivalent with respect to the maxmimal knowledge that can be derived from the corresponding traces.

The presented notions of low equivalences are parametrized by two different kinds of initial key knowledge. As mentioned before, this is as the two runs of the server that are related by the non-interference notion need to be formulated with respect to different secret key knowledge sets. This is necessary to vary the secret in the two runs and at the same time require a maximal attacker knowledge as it is done in Definition 5.

D. Practicability of Security Definitions

Combining the two presented security properties yields a very strong notion of security that exceeds existing synergies between the two definitions: While strong key secrecy alone does not give any guarantees on what the server might leak about confidentially transmitted data, enforcing strong confidential data secrecy alone would already imply a weak form of key secrecy. If a server would leak the secret key used for encrypting the connection as a whole then also connection secrecy would be trivially broken (given that the server logic does not react purely uniformly to all kind of input). However, this does not prevent a server from partially leaking a key: A securely encrypting server that simply leaks its encryption key bitwise with each message would satisfy Definition 6 as keys are treated symbolically the underlying model. By additionally demanding a server to satisfy strong secrecy, this gap in the model closed.

Still, it remains to be discussed whether the presented security notions are achievable in practice. The presented properties are pretty strong in the sense that they do not even allow for (partial) leakage of confidential data by the server's communication structure. In theory, a server could leak confidential data by its response pattern or termination behavior. While it seems reasonable that the communication structure of a server should not depend on a cryptographic key used for encryption, for the case of strong confidential data secrecy this is less obvious as this would require the whole

server logic not to expose any such implicit information leaks. Still, requiring a uniform termination behavior can be a reasonable assumption for web servers as those should generally be assumed to be non-terminating. In contrast, requiring uniform response patterns might be more controversial. Depending on the level of abstraction considered, input-dependent responses of different data size would e.g. become visible by the number of packages transmitted over the network. This could motivate the study of inherently weaker security properties that however we leave for future work.

VII. A Secure Encrypting Middlebox

In order to showcase the feasibility of the presented security notions, we constructed a simple encrypting middlebox, which implements the most basic symmetric encryption protocol (that can be thought as an abstraction of the TLS record layer without authentication), and proved it to satisfy strong key secrecy and strong confidential data secrecy when composed with a large class of web servers.

The middlebox is initialized with a symmetric key k, that was previously shared with the single honest client, and is not known by anyone else. It encrypts every message m that the server sends using the key k and a fresh initial vector r, and then forwards it over the network. Whenever the middlebox receives a message, it tries to decrypt it using key k, and if the decryption succeeds, it forwards the decrypted message to the server. If not, it silently drops the message.

An inductive definition of the middlebox traces written in Coq is shown in Figure 13: At the moment we assume that the

```
Inductive is_middlebox_trace (k:key) : trace → trace → Prop
      :=
| m_empty:
    is_middlebox_trace k [] []
| m_send: ∀ r m tr mtr,
    is_middlebox_trace k tr mtr →
    is_middlebox_trace k (send m :: tr)
                         (send (enc k m r) :: mtr)
| m_recv: ∀ r m tr mtr,
      is_middlebox_trace k tr mtr →
      is_middlebox_trace k (recv m :: tr)
                           (recv (enc k m r) :: mtr)
| m_recv_unenc: ∀ m tr mtr,
      is_middlebox_trace k tr mtr →
      encrypted_recv_message k m = false →
      is_middlebox_trace k tr (recv m :: mtr).
```

Fig. 13. A trace-based specification of the middlebox. Note that encrypted_recv_message k m is a function that returns whether a ciphertext m was encrypted with a specific key k.

server is interacting with only one honest client, with which the middlebox shares the encryption key k.

As the middlebox encrypts messages sent by the server one by one, it is possible for information to be leaked from the messaging patterns of the server. For example, an insecure server, could send the same message N times, when it receives the value N. The middlebox cannot protect the server from this kind of information leakage, therefore we have to require that the the server satisfies *structural indistinguishability*, i.e. the message patterns of the server do not depend on the confidential values that it exchanges with its client.

We prove that for any server satisfying *structural indistinguishability*, the composition of the middlebox and the server satisfies strong key secrecy and strong confidential data secrecy in the presence of a semi-adaptive preplaying attacker (see Section V-B).

We have formalized the presented security notions and the case study in Coq [18] and we are in the process of finalizing the mechanized proofs about the middlebox satisfying strong key secrecy and strong confidential data secrecy.

VIII. Related Work

A. Cryptographically Masked flows

The notion of cryptographically masked flows was first introduced by [5]. This work focuses on a meaningful non-interference based security notion for (interactive) programs that contain cryptographic primitives (namely encryption and decryption operations) and shows how to enforce this notion using a security type system. In particular, this work identifies the problem of occlusion, that specifically arises in that setting, and addresses it by using a possibilistic non-interference notion and by extending the similarity relation of encrypted messages. Even though this work considers interactive programs, it does not account for key leakage.

Extending [5], Askarov et al. [6] study cryptographically masked flows in the presence of key leakage for non-interactive programs. In contrast to [5], this work uses a knowledge-based security notion and relaxes it to account for declassification and key release. In particular they show how to express declassification in terms of key release in their framework. However, one should be careful not to confuse the knowledge notion used in this work with our notion of key knowledge. Knowledge-based security definitions simply present an alternative to classical non-interference based definitions. When studying the impact of key release, [6] need to additionally introduce (similar to our work) an explicit notion of key knowledge and derivability.

B. Knowledge-based Security

Following the line of [5], there has been a lot of work on knowledge-based security [4], [7], [11], [19]. They define knowledge as the uncertainty of an observer about the possible values of confidential input, and confidentiality in terms of reducing this uncertainty, also considering the interactive setting. However, to our knowledge, none of these works considers cryptographically masked flows.

More specifically, Askarov et al. [4] study different attacker notions in the presence of dynamically changing policies. They are not concerned with cryptographic attacker capabilities (e.g. secret guessing), but rather focus on the interplay between policy changes and the history of events that an attacker might refer to, in order to refine their knowledge.

Recently, there has been work in applying knowledge-based reasoning in the web setting, focusing on interactive browsers [19]. In particular, they extend the work on declassification by Askarov et al. [6] by defining a security notion

for web browsers that accounts for dynamic script's behaviors such as the creation of new DOM elements.

C. Non constructive Attacker Definitions

There has been previous work in specifying attacker capabilities in a non-constructive way [8]–[10], although with a different scope. Similarly to us, they don't explicitly specify what the attacker can do, but they specify what the attacker model cannot violate, therefore considering the greatest attacker that does not violate a set of properties. Initially they only targeted reachability properties [8], but they then extended their investigation to equivalence properties, such as strong secrecy [9].

However, the difference from out work is that they focus on getting computational security guarantees from proofs in the symbolic setting. The properties that they restrict the attacker with, are related to cryptographic assumptions, therefore giving them computational security guarantees for free, by proving a protocol secure in that symbolic setting.

D. Formal Protocol Analysis

Cryptographic protocols are often modeled using process calculi (spi-calculus [2] , applied pi calculus [1]), or rewriting logic [21]. In both these approaches, attackers' actions are limited by their knowledge. There restrictions are enforced using different formalisms, most notably the notion of active substitutions [1], [2], which captures the restrictions in the semantics of the respective calculus.

Related to our work, Bertolotti et al. [12] give an efficient way for explicitely representing the attacker's knowledge in the spi calculus. They define a set of natural deduction rules resembling the rewrite rules we present here and show these rules to destruct a set of messages into a normal form that characterizes the minimal knowledge that an attacker needed to construct all data that they exposed in an attack trace.

In general, there is a tremendous amount of work on formal cryptographic protocol analysis for equivalence-based properties which we cannot cover here due to space constraints, hence we refer the reader to the survey by Delaune et al. [16] for an extensive overview of the field.

E. Security Formalization using Proof Assistants

There has been increasing interest during the recent years to formalize security in proof assistants such as Coq and Isabelle. Besson et al. [13] refine existing monitors for non-interference with a notion of attacker knowledge in order to make them more precise. They show that their monitor is sound regarding to a knowledge-based non-interference notion (extended from [6]) and formalize this proof together with the notion of attacker knowledge in Coq.

Van Den Berghe et al. [24] specify a Coq model for designing software systems by composing separate processes in a network of components to facilitate security reasoning. They also provide an explicit representation of attacker knowledge that however differs from our inductive-predicate based approach. They only consider application-specific trace properties as security properties.

Finally, Kanav et al. [17] specify a conference management system in Isabelle [20], and verify that it satisfies several confidentiality properties. They account for declassification by extending Sutherland's Non-deducibility [23].

IX. Discussion and Limitations

The domain of information flow control has been extensively studied in throughout the years resulting in an overwhelming quantity of related work. Even though we put big efforts in assessing the existing literature, we would be thankful for any feedback on the novelty of our approach and related work that we might have missed.

The setting that we present in this work is subject to major simplifications: The cryptographic primitives that we consider are restricted to symmetric key encryption. In particular we do not deal with public key cryptography, nonces, or hash functions as might required for modeling real-world encryption layers. This simplification is possible as we assume a pre-established session between the server and an honest client in that each of the parties already holds the secret shared key. This also exempts us from modeling key generation and key exchange. By restricting to a single client setting we do not need to consider key management on the server side. In particular, we don't need to to place any assumptions on how the server associates its internal client representation with the cryptographic keys used for encrypting the corresponding connections.

X. Conclusion

We have presented security properties for interactive servers with encryption components in the presence of key leakage. To our knowledge, we are the first to address security properties in this setting. In addition, we have proposed non constructive characterizations of attacker capabilities, as a general alternative to constructively specifying an attacker by their possible actions. The strongest of those, named "semi-adaptive" preplaying attacker, can trick honest users into sending arbitrary messages, as well as use keys that were observed during the interaction with the server. Finally, we have proved that a simple encrypting middlebox satisfies the proposed security properties in the presence of a "semi-adaptive" attacker. We are in the process of formalizing all the above in Coq.

Future Work: In the future we would like to overcome some of the aforementioned limitations. In particular we would like to account for key and nonce generation and study the impact of these primitive operations on the characterization of attacker capabilities. Building on top of that, we would like to model authentication and key exchange protocols as needed for establishing a secure connection.

Furthermore, we plan to extend our definitions to allow for modeling multiple clients. In this setting it could also be of interest to refine the client model by associating different attacker capabilities to the secret keys which represent honest clients. Following this idea, it would also be interesting to study possible notions of purposeful declassification in the setting of a (web) server.

References

[1] Martín Abadi, Bruno Blanchet, and Cédric Fournet. The applied pi calculus: Mobile values, new names, and secure communication. *J. ACM*, 65(1):1:1–1:41, October 2017.

[2] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1 – 70, 1999.

[3] Martn Abadi. Security protocols and their properties. In *Foundations of Secure Computation, NATO Science Series*, pages 39–60. IOS Press, 2000.

[4] Aslan Askarov and Stephen Chong. Learning is change in knowledge: Knowledge-based security for dynamic policies. In *2012 IEEE 25th Computer Security Foundations Symposium*, pages 308–322. IEEE, 2012.

[5] Aslan Askarov, Daniel Hedin, and Andrei Sabelfeld. Cryptographically-masked flows. In *International Static Analysis Symposium*, pages 353–369. Springer, 2006.

[6] Aslan Askarov and Andrei Sabelfeld. Gradual release: Unifying declassification, encryption and key release policies. In *2007 IEEE Symposium on Security and Privacy (SP'07)*, pages 207–221. IEEE, 2007.

[7] Musard Balliu. A logic for information flow analysis of distributed programs. In *Nordic Conference on Secure IT Systems*, pages 84–99. Springer, 2013.

[8] Gergei Bana and Hubert Comon-Lundh. Towards unconditional soundness: Computationally complete symbolic attacker. In Pierpaolo Degano and Joshua D. Guttman, editors, *Principles of Security and Trust*, pages 189–208, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[9] Gergei Bana and Hubert Comon-Lundh. A computationally complete symbolic attacker for equivalence properties. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 609–620, New York, NY, USA, 2014. ACM.

[10] Gergei Bana, Koji Hasebe, and Mitsuhiro Okada. Computationally complete symbolic attacker and key exchange. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 1231–1246, New York, NY, USA, 2013. ACM.

[11] Anindya Banerjee, David A Naumann, and Stan Rosenberg. Expressive declassification policies and modular static enforcement. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 339–353. IEEE, 2008.

[12] Ivan Cibrario Bertolotti, Luca Durante, Riccardo Sisto, and Adriano Valenzano. Efficient representation of the attackers knowledge in cryptographic protocols analysis. *Formal Aspects of Computing*, 20(3):303–348, 2008.

[13] Frédéric Besson, Nataliia Bielova, and Thomas Jensen. Hybrid monitoring of attacker knowledge. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 225–238. IEEE, 2016.

[14] Aaron Bohannon, Benjamin C Pierce, Vilhelm Sjöberg, Stephanie Weirich, and Steve Zdancewic. Reactive noninterference. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 79–90. ACM, 2009.

[15] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, September 2010.

[16] Stéphanie Delaune and Lucca Hirschi. A survey of symbolic methods for establishing equivalence-based properties in cryptographic protocols. *Journal of Logical and Algebraic Methods in Programming*, 87:127–144, 2017.

[17] Sudeep Kanav, Peter Lammich, and Andrei Popescu. A conference management system with verified document confidentiality. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification*, pages 167–183, Cham, 2014. Springer International Publishing.

[18] Coq development team. *The Coq proof assistant reference manual*. LogiCal Project, 2018. Version 8.8.1.

[19] McKenna McCall, Hengruo Zhang, and Limin Jia. Knowledge-based security of dynamic secrets for reactive programs. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 175–188. IEEE, 2018.

[20] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*, volume 2283. Springer Science & Business Media, 2002.

[21] Sonia Santiago, Santiago Escobar, Catherine Meadows, and José Meseguer. A formal definition of protocol indistinguishability and its verification using maude-npa. In *International Workshop on Security and Trust Management*, pages 162–177. Springer, 2014.

[22] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, July 1948.

[23] David Sutherland. A model of information. In *Proceedings of the 9th national computer security conference*, volume 247, pages 175–183. Washington, DC, 1986.

[24] Alexander van Den Berghe, Koen Yskout, Wouter Joosen, and Riccardo Scandariato. A model for provably secure software design. In *Proceedings of the 5th International FME Workshop on Formal Methods in Software Engineering*, pages 3–9. IEEE Press, 2017.