InfoLeak: Scheduling-based Information Leakage

Tsvetoslava Vateva-Gurova*, Salman Manzoor*, Yennun Huang† and Neeraj Suri*
*TU Darmstadt, CS Department, Germany
{vateva, salman, suri}@deeds.informatik.tu-darmstadt.de
†Research Center for Information Technology Innovation, Academia Sinica, Taiwan
{yennunhuang}@citi.sinica.edu.tw

***Abstract*—Covert- and side-channel attacks, typically enabled by the usage of shared resources, pose a serious threat to complex systems such as the Cloud. While their exploitation in the real world depends on properties of the execution environment (e.g., scheduling), the explicit consideration of these factors is often neglected.**

This paper introduces InfoLeak, an information leakage model that establishes the crucial role of the scheduler for exploiting core-private caches as covert channels. We show, formally and empirically, how the availability of these channels and the corresponding attack feasibility are affected by scheduling. Moreover, our model allows security experts to assess the related threat, posed by core-private cache covert channels for a particular system by considering solely the scheduling information. To validate the utility of InfoLeak, we deploy a covert-channel attack and correlate its success ratio to the scheduling of the attacker processes in the target system. We demonstrate the applicability of the InfoLeak model for analyzing the scheduling information for possible information leakage and also provide an example on its usage.

***Index Terms*—information leakage model, scheduling, side-channel attacks, covert-channel attacks, feasibility**

1. Introduction

The first studies related to covert-channel attacks (CCAs) and side-channel attacks (SCAs) go back decades [1], [2]. However, with the advent of Cloud computing their relevance has even increased due to the inherent resource sharing among different services and customers in the Cloud [3], [4]. The access to shared resources (e.g., the L1 cache targeted in this paper) is a prerequisite for CCAs and SCAs, and the shared resource being exploited is termed as the covert channel (CC) for the rest of the paper. As the CCs are not explicitly designed to be used for communication, SCAs and CCAs are hard to be detected using traditional intrusion detection approaches.

As background, Millen [5] defined four major areas of research in the field of CCs, namely: (i) explaining, (ii) finding, (iii) measuring, and (iv) mitigating CCs. These areas have been extensively explored where the research activities focus on finding new ways to exploit a system through a CC (e.g., [3], [6]), increasing the bandwidth of the covert-channel communication (e.g., [7], [8]), or protecting the system from such exploits (e.g., [9], [10], [11], [12], [13]). As a CC fundamentally deals with information, developing accurate models of information leakage is of practical relevance as such models can assist security experts in analyzing the robustness of their systems against SCAs and CCAs. Thus, developing a CC information leakage model that can explicitly handle the considerations of the execution environment, such as scheduling, is the focus of this paper.

1.1. Problem Statement

Although a variety of formal models for CCAs [14], [15], [16] and SCAs [17], [18] exist in literature, they primarily focus on the way the information is leaked for specific attacker scenarios, and also on the achievable bandwidth. However, there are various properties of the execution environment (e.g., CPU scheduling, shared memory, etc.) that can influence the feasibility of the attack, along with the previously mentioned characteristics e.g., bandwidth [9], [19], [20], [7], [8], [4], [3]. The impact of the execution environment has been partially studied [20], [9], but the execution environment's properties are not incorporated in any of the existing information leakage models. This consequently impedes the systematic analysis of the threat posed by the CC. Of particular interest is the role of the CPU scheduler and its impact on the core-private cache CC[1] exploitability.

Scheduling, as a fundamental resource allocation schema, directly affects the synchronization for the accesses to the CC and the channel's availability, especially for attacks using the core-private cache. As proper synchronization is a prerequisite for a successful SCA or CCA, scheduling has an effect on the feasibility and the bandwidth of the covert-channel communication, as shown in [1], [9], [20]. However, the lack of an information leakage model that explicitly considers the impact of the scheduling exacerbates conducting a comprehensive system security covert channel analysis, as it omits this crucial execution feasibility consideration. Despite the merits of the existing information models, they currently do not provide the means to (a) assess

1. "CC" and "channel" are used interchangeably throughout the paper.

whether information leakage can possibly transpire in the system, or (b) project its impact based on the available scheduling information as a feasibility metric.

1.2. Contributions & Paper Organization

We develop an information leakage model, termed as InfoLeak. To the best of our knowledge, InfoLeak is the first formal model that considers the effect of CPU scheduling on the consequent feasibility of SCAs and CCAs exploiting the core-private cache. InfoLeak facilitates the systematic reasoning regarding potential SCAs and CCAs threats, and enables system designers to use the scheduling information for assessing the information leakage feasibility or, conduct SCA/CCA post-mortem analysis. Studying the influence of the CPU scheduling is a key attribute behind the security analysis, as the CPU scheduling can result in lack of synchronization for the access to the channel and make its exploit impossible. However, the direct integration of the scheduling algorithm into the security analysis is not straightforward due to the involved indeterminism. Thus, InfoLeak aids security analysis by ascertaining the correlation between the CPU scheduling traces which are available and the success of a covert-channel exploit.

InfoLeak can help identify potential security breaches related to the exploitation of the core-private cache as a CC by using only the CPU scheduling information. It can serve as an indicator for the possible presence of a covert-channel attacker without neglecting the noise stemming from the workload on the system caused by other processes. A threshold for the amount of possibly transmitted or eavesdropped information can be considered as a measure for the feasibility of the attack, and used to raise an alert. This can help detecting possible leakage and triggering respective actions (e.g., informing the parties that might have been attacked or de-scheduling the suspected process) or, conducting a post-mortem analysis of a system being under attack.

To validate the proposed information leakage model, we implemented and deployed a CCA and correlated the success of the covert-channel communication with the scheduling information for the attacker's processes. In addition, we demonstrate the utility of InfoLeak on a synthetic example resembling a well-known SCA by showing how InfoLeak can assist in identifying the given attack.

Paper Contributions: Overall, our contributions are: (i) developing an information leakage model for core-private cache-based SCAs and CCAs that explicitly considers the scheduler's influence on the attacks and allows for assessing their feasibility given the system's scheduling information in Section 4, (ii) demonstrating the utility of the proposed model by applying it to a state-of-the-art example in Section 5, and (iii) empirically validating the proposed model by applying it to a L1 cache-based CCA and using the available scheduling traces in Section 6.

Key Findings: The experimental results show a strong positive correlation of 0.8 between the successful transmission of bits over the core-private cache CC, and a favourable scheduling in the context of CCAs considering the available scheduling traces in a setup with CPU load of 40%, and a strong positive correlation between the successful transmission of 0.44 in a setup with 80% CPU load. This ascertains the usefulness of the usage of scheduling traces as a feasibility metric for cache-based covert-channel exploits, and as an upper bound of possible information leakage.

2. Related Work

The first part of this section reviews the contemporary SCAs/CCAs exploiting the cache as a CC, and presents formal modeling approaches in the SCA/CCA area. Then, works considering the CPU scheduling effect on the feasibility of these attacks are discussed.

2.1. Side- and Covert-Channel Approaches

A variety of SCAs and CCAs exploiting the cache as a CC have been demonstrated in both virtualized and non-virtualized environments. These attacks usually apply either the Prime+Probe technique [4], [3], [7], or the Flush+Reload technique [21], [22], [23] to acquire secret data.

The Flush+Reload attacks typically rely on the existence of shared memory between the victim and an attacker (e.g., a shared library). Although these attacks represent a powerful way to leak information, they are not in the focus of our paper, as they can be prevented by disabling the possibility that the victim and the attacker share memory. This preventive measure has already been considered in the real world and applied in practise (cf., [24]).

The attacks exploiting the Prime+Probe technique typically employ a process accessing the whole cache repeatedly and measuring the accesses to the different cache parts. The attacks considering the core-private caches are the main concern of this paper. Important works, employing this technique, were proposed in [4], [7], exploiting the L1 and L2 caches, respectively.

2.2. Information Theory and Covert Channels

Shannon pioneered applying information theory to analyze communication flows and provided [25] an extensive mathematical model for communication over discrete communication channels. Lampson characterised a number of leakage sources in [2], and classified the leakage channels into storage-based, legitimate and covert. His work proposed confinement rules to prevent leakage, but they are restrictive to apply in practice.

Kemmemer [26] investigated information flows through illegitimate channels and proposed a methodology to increase system's security by finding all possible CCs through a shared resource matrix. He introduced criteria for identifying both storage and timing channels. In [27], Wray argued that this classification of CCs to (i) storage and (ii) timing, as presented in [26] is not representative enough. Instead, he considered a CC as a channel with storage and

timing attributes, and defined a clock as being characterized by sequences of events which can be distinguished from each another. As both approaches require the exhaustive enumeration of all shared resources or clocks, they are not practically feasible. The CC detection depends on the level of granularity to which the system is analyzed, as a system level information leakage model is lacking.

The Bucket model for microarchitectural information channels was presented in [15] aiming to cover all the characteristics of the microarchitectural channels and to detect attackers. However, the proposed model suggests a different attacker model than the commonly used one in CCAs. In [15] the communicating entities are not adversaries of the system. Instead, a third party is assumed to eavesdrop their communication through the CC. This work also does not explicitly consider the system's operational parameters.

Lee et al. also focused on modeling CCs and estimating CC capacity in [28], [29]. The authors asserted that covert-channel communication is intrinsically not synchronous. In [28], [29], the CC is represented as an insertion-deletion channel to reflect upon the communication degradation due to the lack of synchronization. We consider this observation as crucial for the cache-based CC analysis, as the synchronization errors in the CC are not negligible. Thus, we consider both works as a starting point for our analysis.

2.3. The Effect of Execution Environment

Hu [30] noted the importance of synchronization for the covert-channel exploits, and proposed a method for significantly decreasing the capacity of the CC by using fuzzy time system clocks. His approach disabled proper synchronization between the communicating parties by ensuring that the receiver process does not have access to an accurate reference clock. Hu experimentally demonstrated that the capacity of the channel is reduced, but his approach prevents processes from accessing an accurate time source which might be needed.

In [31] Gray noticed that the channel capacity depends on the assumptions about the execution environment. The author modeled system's and environmental's probabilistic behavior independently and expressed channel's capacity in information theoretical terms. He considered the system as a finite set of states along with a set of communication channels providing interface to the external environment. Gray's work does not focus on how to use scheduling information to obtain information about possible covert communication, Gray's distinction between system under consideration and the environment in which the system is executing is very important for the covert- and side-channel analysis. Gray used this distinction in [14] to obtain an upper bound on the capacity of the CC under fuzzy time as introduced by Hu in [30].

Works considering the synchronization (i.e., the scheduling effect) as a challenge or even feasibility aspect for conducting SCAs and CCAs were published in e.g., [4], [6], [20]. The authors of [4] describe how they manage to synchronize and overwhelm this challenge by using interprocess interrupts. The scheduling policy was also used as a defense mechanism against cache-based SCAs in the research described in [9]. Although works related to the impact of scheduling on the accesses to the CC exist, there is no comprehensive, systematic analysis on the topic that might be applied on varied systems to analyze the covert-channel threat using the scheduling information.

3. Covert Channels: System and Attacker Models

This section presents InfoLeak's system model along with the CCA's and SCA's attacker models focusing on the core-private cache as a CC.

3.1. System Model

Both SCAs and CCAs compromise the confidentiality of a system i.e., confidential information is conveyed to a malicious party. Often the core-private caches get misused as a CC. The cache-based SCAs and CCAs are relevant for both virtualized and non-virtualized environments, though in the virtualized environment an additional layer is assumed to provide isolation among the virtual machines. In practice, it is solely an additional source of noise in the CC.

We propose an information leakage model that is applicable to both environments by abstracting from the software layers. Our system model is thus narrowed down to focus only on the core-private cache as a CC for communication between processes running on the system, as shown in Fig. 1. The cache is small memory enabling fast access to frequently used data. The processors are typically characterized by a hierarchy of caches where the last level caches (LLC or L3) are the slowest and largest, and the core-private level 1 (L1) caches are the smallest and fastest. Depending on where the requested data is stored (cache level or main memory), accessing it is faster or slower. If the data is available in the cache, a cache hit takes place. In case of a cache miss, the data has to be fetched from the main memory and copied into the cache. The timing difference between a cache miss and a cache hit is often a premise for the exploits of the cache as a CC. We are focusing on CCAs/SCAs where the information is transmitted through the footprint a sending process (or a victim) leaves on the core-private cache. The sending and receiving processes can be any arbitrary non-privileged processes using the same core-private cache.

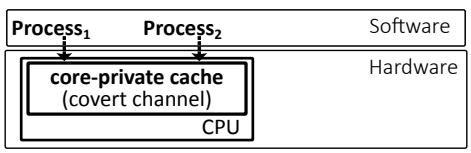


Figure 1. System model.

3.2. Attacker Model: The Covert-Channel Exploits

To illustrate the similarities between SCAs and CCAs, we utilize the channel notion introduced by Shannon [25], and define message, information source, transmitter, signal, noise source, receiver and destination for each attack type.

An SCA involves a victim and an attacker, whereas a CCA is characterized by two cooperating attackers – a sender and a receiver. As depicted in Fig. 2 (using red color), the SCA victim (i.e., information source) unintentionally encodes secret data into the channel. This data is the message being transmitted. In a CCA (cf., Fig. 2 using blue color), the message is the information the sender (i.e., information source) intentionally transmits to the receiver. The transmitter encodes the message into the channel. This role is taken by the victim's (SCA) or sender's (CCA) process leaving its footprint into the core-private cache. Intuitively, for both attack types, the signal is the respective footprint. The receiver is the attacker's process trying to analyze the sampled cache footprint of the victim's or sender's process. The destination is the attacker eavesdropping on the victim's data or receiving sender's data. Noise source can be any process other than the victim's/sender's process that interacts with the core-private cache.

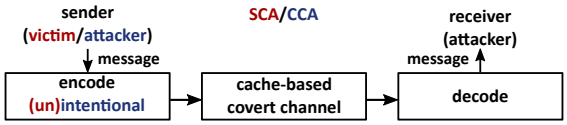


Figure 2. Side- and covert-channel attack distinction.

For simplicity, we refer to both the SCA's victim process and the CCA's sender as the *sender*, and to the SCA's attacker process and the CCA's receiver as the *receiver*. Without loss of generality, we model the communication as a one-way process where the information is only conveyed from the sender to the receiver. The sender is not restricted to a particular user of the system and does not need any special privileges. Thus, the same user might be both sender or receiver at different points in time. As the core-private cache is used in the same way to transmit information in both CCAs and SCAs, a proper synchronization is needed in both cases. Ideally, the receiver accesses the channel directly after the sender. As InfoLeak focuses on the channel itself and the synchronization issues related to its usage, it is applicable to both SCAs and CCAs.

The focus of this paper is on the core-private caches as a CC. As these caches are relatively small in size, their content is often overwritten. Thus, we consider the studied channel as a discrete memoryless communication channel.[2] where the output probability distribution only depends on the current channel input. Moreover, as not each input symbol to the CC is received due to synchronization errors, we model the core-private cache channel as a discrete memoryless channel with synchronization errors.

2. This parallels the established information leakage models [28], [32] that model a CC as a memoryless synchronous channel with or without noise.

4. Modeling the Cache Covert Channel

This section presents our information leakage model, called InfoLeak. InfoLeak formally describes how the covert channel is exploited, and how its exploitation is affected by the CPU scheduling. We adopt the established definition of a covert channel, and consider it as a communication channel stemming from the usage of shared resources that is not intended to be used as such by the designers of the system and does not require special privileges to be used. As mentioned earlier, our focus in on the core-private cache as a covert channel.

4.1. Covert Channel Users

InfoLeak defines three user categories that interact with the covert channel, namely the *sender*, the *receiver* and *other processes*, abbreviated respectively as *S*, *R* and *O*. Each of the user categories can access the core-private cache and change its state. While *S* accesses the channel to send information (on purpose as a CCA attacker or unintentionally as an SCA victim), *O* is not necessarily aware of the existence of this communication channel and encodes only noise into it. *R* accesses the channel on purpose to obtain the information sent by *S*, but similarly to *O*, encodes only noise into the channel.

4.2. Covert Channel States

Through their actions (i.e., cache accesses), the cache users change the state of the covert channel. Hence, we differentiate between the cache states O_i (observation$_i$) that correspond to cache patterns which encode meaningful information to R, and a state U for undefined. U can be considered as the state resulting from noise perturbations in the channel. Noise can occur, e.g., if S had encoded information into the channel but this information has been overwritten by another process (i.e., O) before R could decode it. In such a case, S and R are not synchronized well, possibly due to the scheduling effect. As an example, a CCA scenario can be characterized by two predefined cache footprints representing bits 0 and 1 being transmitted. Then, the states the cache exhibits are O_1 and O_2 (mapped to bits 0 and 1), and the undefined state U.

4.3. Covert Channel Interactions

To model the interactions with the channel, we employ a non-deterministic finite automaton (NFA) A. The states of the channel are represented as states in the NFA and the interactions of the users with that channel trigger transitions in the NFA. $A = (Q, \Sigma, \Delta, q_0, F)$ consists of:

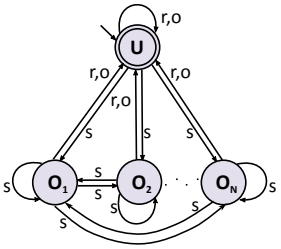
- a finite set of states $Q = Ob \cup \{U\}$, where $Ob = \cup_N^{i=1}\{O_i\}$ for $N \in \mathbb{N}$;
- a finite set of input symbols $\Sigma = \{s, r, o\}$;
- an initial state $q_0 = U$;
- a set of accepted states $F = \{U\}$;

TABLE 1. NFA A TRANSITION FUNCTION.

		Input Symbol		
		s	r	o
State	O_i	Ob	$\{U\}$	$\{U\}$
	U	Ob	$\{U\}$	$\{U\}$

- a transition function $\Delta : (Q \times \Sigma) \rightarrow 2^Q$, as defined in Table 1.

The set of input symbols Σ corresponds to the user responsible for triggering a transition. *S*, *R* and *O* trigger transitions s, r, and o, respectively. We assume that S always encodes information into the cache. Hence, *S* always changes the state of the channel to O_i with its transitions, as depicted in the NFA in Fig. 3 (cf., s-transitions). For example, in a CCA, S can send one bit or a symbol at a time to R by leaving recognizable cache footprints. The r-transitions in the NFA in Fig. 3 illustrate the channel state changes induced by *R*, aiming to decode the encoded information. If *S* and *R* are synchronized, R causes a transition from a state O_i to the state U. This corresponds to *R* receiving information and producing noise in the channel through its decoding operations. As the operation of receiving information is then successful, U is a part of the accepted states F. Since *R*'s interactions with the channel always correspond to noise perturbations, the r-transitions lead to the undefined state U, as shown in the NFA in Fig. 3.



1	$r, o \rightarrow ro$ $r, r \rightarrow rr$ $r, Z \rightarrow rZ$ $o, o \rightarrow oo$ $o, r \rightarrow or$ $o, Z \rightarrow oZ$
2	**r, s $\rightarrow \varepsilon$** $o, s \rightarrow os$
3	$s, o \rightarrow so$ $s, r \rightarrow sr$ $s, Z \rightarrow sZ$
4	$s, s \rightarrow so$

Figure 3. NFA A (left) and P's stack (right).

It can be noticed that the transitions triggered by *O* (o-transitions in Fig. 3) are the same as those triggered by *R*. Both *R* and *O* produce only noise into the covert channel and, their transitions lead to state U. Still, for the further analysis, we need to differentiate between *O* and *R* due to *R*'s intentional usage of the channel in contrast to *O*'s unintentional interaction with the channel. To avoid complicating the model unnecessarily, we assume that the cache is in an undefined state before the attack. Thus, U is the initial state of A.

4.4. Scheduling Effect on the Cache Covert-Channel Exploits

A prerequisite for the success of cache-based CCAs and SCAs is the synchronization between S and R. Thus, the effect of the CPU scheduling determining which process is running on which core is crucial for the CCAs or SCAs success especially when using the core-private cache. To model this scheduling effect, we extend the NFA A to a non-deterministic pushdown automaton (PDA) $P = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ by adding a stack. Hence, we define additionally an initial stack symbol, a stack alphabet and a new transition function, as follows.

- a stack alphabet $\Gamma = \{Z\} \cup \{s, r, o\}$;
- a transition function $\delta : (Q \times \Sigma \times \Gamma_\varepsilon) \rightarrow 2^{Q \times \Gamma_\varepsilon}$, defined in Table 2;
- an initial stack symbol Z;

TABLE 2. PDA P TRANSITION FUNCTION.

Trans.#	Transition
1	$\delta(U, s, Z) = \cup_N^{i=1} \{(O_i, sZ)\}$
2	$\delta(O_i, s, s) = \cup_N^{i=1} \{(O_i, so)\}$
3	$\delta(U, s, r) = \cup_N^{i=1} \{(O_i, sr)\}$
4	$\delta(U, s, o) = \cup_N^{i=1} \{(O_i, so)\}$
5	$\delta(U, r, Z) = \{(U, rZ)\}$
6	$\boldsymbol{\delta(O_i, r, s) = \{(U, \varepsilon)\}}$
7	$\delta(U, r, r) = \{(U, rr)\}$
8	$\delta(U, r, o) = \{(U, ro)\}$
9	$\delta(U, o, Z) = \{(U, oZ)\}$
10	$\delta(O_i, o, s) = \{(U, os)\}$
11	$\delta(U, o, r) = \{(U, or)\}$
12	$\delta(U, o, o) = \{(U, oo)\}$

The finite set of states in P, Q is detailed in Section 4.2. InfoLeak uses ε to denote that the element at the top of the stack is popped, and no new element is pushed onto the stack (i.e., the stack is popped). In the proposed model with each move an input symbol is read, and the element at the top of the stack is read and popped. Then, a string consisting of the popped element and a new element, or ε, is pushed onto the stack, so that the new element is on the top of the stack. P's transition function, (cf., in Table 2), takes a state, an input symbol (no ε-transitions considered) and a stack symbol (which can be ε and denotes the element on the top of the stack) as an input. The triple is mapped to a set of states and a set of stack symbols where the first element denotes the top of the stack (e.g., s in sZ is the element on the top of the stack). Fig. 3 visualizes the changes in the states along with the respective stack operations.

The transition $r, o \rightarrow ro$ in Fig. 3 shows that o is currently on the top of the stack, and r is the input element. The new topmost element in the stack is r while the state U remains unchanged (see the NFA in Fig. 3). This behavior is defined by Transition 8 in Table 2. The only successful transmission of information corresponds to a transition $r, s \rightarrow \varepsilon$ (cf., first row of Transition 2 in Fig. 3 or Transition 6 in Table 2). In this case, we pop the stack. The symbol s is read and popped and no new element is pushed onto the stack. A transition from a state O_i to the state U takes place, as R accesses the channel directly after S.

If an s in the stack is followed by another s (cf., Transition 2 in Table 2), or by an o (cf., Transition 10 in Table 2), the sent information is overwritten, as the cache footprint will be changed before the receiver has analyzed it. Thus, if an s is on the top of the stack, and another s comes, the first s is popped and "substituted" by an so (cf., Transition 2 in Table 2).

The idea behind InfoLeak is to incorporate the order of the interactions with the channel (i.e., CPU scheduling effect) into the model. Thus, anytime *S*, *R* or *O* interacts with the channel, a state change is triggered. InfoLeak's states correspond to the view on the core-private cache from an observer's perspective, and allow for encoding information of different granularity into the channel through the states O_i, as it abstracts from details such as specific footprints. This makes the model applicable to both the SCA and CCA scenarios. For instance, it can be generalized to include cases where symbols instead of single bits are transmitted in a CCA, e.g., by adding a new state O_i in the NFA for each additional symbol and considering a single undefined state U.

An estimate of the bandwidth of the attack can also be inferred by keeping track of the number of successive times the stack has been popped over certain time period. An empty stack indicates that all the sent information was received and there was no contention on the cache with other processes. From a stack of a large size, it can be inferred that either there were too many other processes interacting with the cache, or that *S* and *R* were not synchronized. A threshold can be defined for the number of successive times the stack has been popped and can be used as a measure for the feasibility of CCAs and SCAs.

5. InfoLeak Application Scenario

To show the utility of InfoLeak, we apply it to a synthetic example, resembling a well-know SCA. Then, we provide insight on how InfoLeak can be applied for a post-mortem analysis to investigate covert-channel communication.

5.1. An Example of InfoLeak's Utility

To present an example, we simulate the SCA proposed in [4] by Zhang et al. and model it using InfoLeak. In the described work, the attacker abuses the scheduler to interrupt the victim frequently enough to be able to obtain sufficient observations on victim's cache footprints. The attacker collects vectors of 64 values each representing the timings for accessing the 64 cache sets and maps them to the Square (Sq), Multiply (M) or ModReduce (MR) operations of the square-and-multiply algorithm used for fast exponentiation. Based on these observations, the attacker can extract victim's secret key. To model the attack using InfoLeak, we consider the Sq, M and MR as the channel states O_i in the NFA P, as shown in Section 4. Intuitively, the cache footprints that represent a squaring operation are mapped to the state Sq. Analogously, the states M and MR correspond to the cache footprints representing a multiply and modular reduction operations, respectively. The resulting NFA is shown in Fig. 4.

Ideally, two successive observations on this channel reveal one secret key bit, if the bit is a 0. For revealing a bit 1, the attacker needs four successive observations on the channel directly after the victim. Thus, if the bits of the secret key are normally distributed, the attacker would need

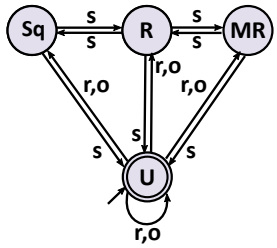


Figure 4. Example SCA.

three successive observations on average to extract one bit. This results in the sequence s-r-s-r-s-r in InfoLeak, i.e. popping the stack three successive times. Having this in mind, a threshold can be set to indicate the feasibility of the SCA in the system. The scheduling information can be analyzed for the successive scheduling of any two processes considering this feasibility threshold. In [4], the frequent interrupts caused by the attacker will be visible in the scheduling information and can be used as an indication that an attack was feasible. It must be noted, that few bits might lead to leaking a secret key if the SCA is combined with another attack. Still, it would be more beneficial for an attacker if the bits are successive. This observation can be used to accordingly adjust the feasibility threshold.

5.2. Post-Analysis of Scheduling Information

InfoLeak can be applied to analyze the scheduling log of the operating system for suspicious cache-based covert communication, as well. The suspicious processes can be represented as an *S* and an *R*, and the rest of the processes – as *O*'s, as described in Section 4. By applying InfoLeak transitions to the scheduling log, we can build a stack of s's r's and o's, where s and r correspond to the transitions changing the cache state triggered by the two suspicious processes. All other processes in the scheduling log represent noise. Considering the number of successive popped s's over certain time period, can give an estimate of the feasible bandwidth of the leakage that might have happened.

While identifying suspicious behavior is a challenging task, certain hints could help uncover it. Suspicious behavior could be, for instance, any process pair consisting of two CPU intensive processes, running (almost) always on the same CPU core or a process issuing frequent inter-process interrupts. In addition, if it is noticeable that two processes try to synchronize their operations, they might also be trying to exploit the cache as a CC.

6. Validation

To empirically validate InfoLeak's utility, we employ a CCA using the L1 cache, and correlate its success ratio to the CPU scheduling of the attacker processes. This section provides details on the attack implementation, the experiments and concludes with a discussion on the results.

6.1. Implementation Details

Our CCA adhere to the Prime+Probe protocol, as described in [33], and consists of a sender and a receiver processes. They communicate with each other by indirectly accessing the L1 cache, and have agreed on the meaning of distinct cache footprints which are used for transmitting the bits 0 and 1. Each of the processes allocates memory of the size of the L1 cache (32KB). Through this memory, the sender and the receiver try to encode and decode the predefined cache footprints representing the bits 0 and 1.

In our implementation, the sender encodes one bit of information at a time into the channel by accessing a subset of the L1 cache. To send a bit, the sender accesses specific parts of the allocated memory that are copied into predefined cache sets. In that way, the sender evicts the receiver's data that had been in this part of the cache, and fetches other data from the main memory or from the higher cache levels to the L1 cache. To decode a message, the receiver accesses the cache through its allocated memory and measures the time for the accesses. Depending on the consumed time, the receiver can assess which cache parts have been evicted by the sender. In that way, the receiver determines the cache footprint and whether the cache state corresponds to bit 0, bit 1 or undefined, as detailed in Section 4.

The conducted attack consists of a training and a testing phase. In the training phase we collect ground truth data regarding the consumed clock cycles when accessing the parts of the cache corresponding to bits 0's and 1's.

To obtain a reliable time reference, we sample the timestamp counter hardware register TSC. As both the compiler and the processor do not guarantee executing the instructions in their original order, we try to enforce them to respect the order. Thus, we use volatile as a memory barrier and the CPUID instruction for serialization. The out-of-order execution was considered also by Timor et al. in [34].

As discussed in [35], conducting a CCA is complicated through the distinction between physical and virtual memory addresses. Our sender and the receiver processes do not have any special privileges, and have a view only on the virtual addresses of their allocated memory blocks. Although this complicates accessing particular cache parts, it is still possible to access specific L1 cache sets by conducting operations with specific parts of the allocated memory blocks.

To ensure that the sender and the receiver use the same L1 cache, they request to be scheduled on the same CPU core through scheduler's *setaffinity* option. We assume that they have agreed on the usage of a specific CPU core in the same way they have agreed on the meaning of the cache footprints.

6.2. Experimental Setup

In the testing phase of our attack, we conducted experiments on a Debian Stretch in noiseless and noisy environments. A CCA is run 100 times to transmit messages of length 1200^3 sent by the sender in varied setups. The receiver tries to decode the sent data by accessing the whole L1 cache 1200 times per experiment. In our setup the measurements are done by an assembly program, and the decoding is conducted offline by scripts once the access times have been collected, so that receiver's decoding does not interfere with or invalidate the measurements, as the L1 cache is very sensible to noise.

Simultaneously, the CPU scheduling information is logged in a file. To acquire this information, we make use of the tracing options provided by */sys/kernel/debug/tracing* in Debian and set the *context switch* event to *enabled*. We parse these logs to represent them as the InfoLeak stack (cf., Section 4), and correlate the attack's success ratio to the scheduling log. To assess the correlation between a successful transmission of information and the scheduling of the receiver's process directly after the sender's process statistically, we employ Pearson's correlation coefficients. As independent and dependent variables, we use the *number of favourable scheduling cases*, i.e., the receiver is scheduled directly after the sender, corresponding to the number of $s, r \rightarrow \varepsilon$ -transitions (cf., Table 2) per experiment, and the *number of successfully transmitted bits* over the channel per experiment, respectively. Furthermore, using InfoLeak we give a capacity estimate of the channel in mathematical terms.

6.3. Empirical Analysis

Noiseless environment.. To minimize the noise in the channel, we dedicate one CPU core solely to the receiver and sender using the $isolcpus$ parameter. For all experiments, only 11 bits on average were not properly decoded. The favourable scheduling cases were 1199 on average with a standard deviation of 0.89. For comparison, the mean value of the successfully transmitted bits is 1189 with a standard deviation of 4.151. Table 3 shows a summary of the results.

TABLE 3. RESULTS SUMMARY - NOISELESS SETUP.

Conducted experiments	100
Receiver operations (avg)	1200
Sender operations (avg)	1200
Receiver actually scheduled (avg)	1202
Sender actually scheduled (avg)	1201
Favourable scheduling cases FSC (avg)	1199
Standard deviation for FSC	0.89
Successfully transmitted bits STB (avg)	1189
Standard deviation for STB	4.151

The relationship between the successfully transmitted bits and the favourable scheduling is visible in Fig. 5. Each point in the linechart represents one experiment. The blue plot depicts the number successfully transmitted bits per experiment, and the red plot illustrates the number of favourable scheduling cases per experiment. Due to the low

3. Chosen for statistically significant coverage and represents 120,000 cases.

noise, the receiver was almost all of the 1200 cases scheduled directly after the sender. The ratio of the successfully transmitted bits is also high due to the lack of contention for the channel and the atomicity of the sender and receiver operations.

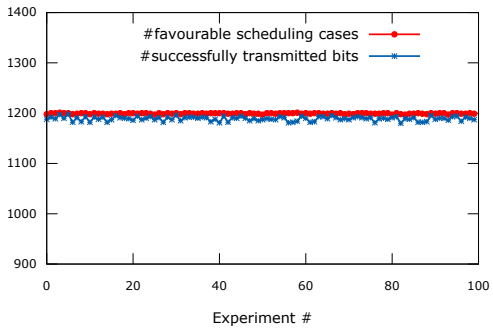


Figure 5. Noiseless setup. Correlation between #favourable scheduling cases (red) and #successfully transmitted bits (blue).

The similarity between the two plots confirms the influence the CPU scheduling has on the CCA's success. The small discrepancy in the values that are slightly visible in the linechart Fig. 5 are due to the few cases in which the send and receive operations were not atomic. The receiver and sender were scheduled 1202 and 1201 times on average, respectively, instead of 1200 times. As there are only few such cases, InfoLeak can still reliably assist in providing an upper bound on the CC capacity. The results are summarized in Table 3.

Noisy environment.. To apply InfoLeak in a more realistic scenario, we conducted 200 experiments in a noisy setup simulating load on the system with the stressing tool stress-ng. We considered background load levels of 40% and 80% for each of the CPU cores.

TABLE 4. RESULTS SUMMARY - NOISY SETUP.

	40%	**80%**
Conducted experiments	100	100
Receiver operations (avg)	1200	1200
Sender operations (avg)	1200	1200
Favourable scheduling cases FSC (avg)	1138	905
Standard deviation for FSC	134	266
Min FSC (over all experiments)	534	255
Max FSC (over all experiments)	1201	1201
Successfully transmitted bits STB (avg)	934	534
Standard deviation for STB	225	200
Min STB (over all experiments)	289	194
Max STB (over all experiments)	1194	1030
Pearson correlation coefficient	0.8	0.44

For the 100 experiments with load of 40%, 934 bits on average were successfully transmitted, and in 1138 cases on average, the scheduling was favourable for the sender and the receiver (average number of $s, r \rightarrow \varepsilon$ - transitions). The standard deviation for the successful bit transmissions is 225, and for the favourable scheduling it is 134.

In the more noisy environment with load of 80%, for the 100 experiments, 535 bits on average were successfully transmitted, and the favourable scheduling cases were 905 on average. The standard deviations are 200 and 266, respectively. The respective minimal and maximal values for successful transmission and favourable scheduling are given in Table 4 along with a summary of the results.

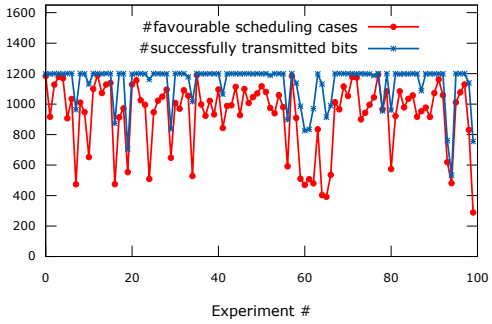


Figure 6. Workload 40%. Correlation between #favourable scheduling cases (red) and #successfully transmitted bits (blue).

The relationship between the number of successfully transmitted bits and the favourable scheduling for both loads is depicted in Fig. 6 and Fig. 7. Each point in the linechart is a representation of a single experiment consisting of 1200 receiver operations. The blue plots show the number of successfully transmitted bits for each of the 100 conducted experiments per linechart. The red plots in the figures show the number of favourable scheduling cases per experiment. As noticeable from the similar trends of the linecharts, there is a possitive correlation between the scheduling and the success of the attack. Calculating the Pearson's correlation coefficient we obtain a very strong positive correlation of 0.8, and a strong positive correlation of 0.44 for the 40%- and the 80%- setups, respectively.

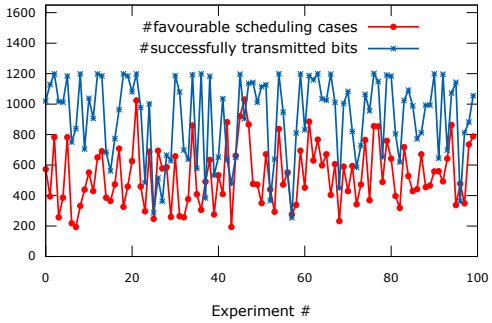


Figure 7. Workload 80%. Correlation between #favourable scheduling cases (red) and #successfully transmitted bits (blue).

The results listed in Table 4 show that the more noise in the CC, the weaker the correlation between the successful transmission and the favourable scheduling. This relies on the atomicity assumption of the sender and receiver operations that InfoLeak considers. In the more noisy setup, the

sender is scheduled only 982 times on average, and the receiver is scheduled 1178 times on average which affects the successful transmission. In such cases, although the receiver is scheduled directly after the sender, the transmitted bits cannot be properly decoded. However, the favourable scheduling cases can still be used as a feasibility metric of the attack, as the scheduling traces nevertheless provide an upper bound on the feasible capacity of the channel.

6.4. Summary of Results

In the validation, we demonstrate that using InfoLeak it is possible to derive an upper bound of the capacity of the L1 cache channel between two processes, as there is a correlation between the number of consequent occurrences of the sender and receiver to the success ratio of the attack. Thus, a capacity estimate can be obtained by counting the number of the $s, r \rightarrow \varepsilon$ - transitions.

We compute the capacity (C) of the channel used for the experiments theoretically using the mutual information (I) between the sender and the receiver processes. Considering uniform probability distribution of the input, we calculate C, as shown in Eq. 1 and described in [25]. The input and output probabilities, $p(x)$ and $p(y)$, respectively, are employed in the capacity computation.

$$C = I = \sum_{x} \sum_{y} p(x,y) log(\frac{p(x,y)}{p(x)p(y)}) \qquad (1)$$

In our scenario, the input values are bits 0 and 1, and the output values are 0, 1, and U. Assuming that the sender sends bits 0 and 1 with equal probabilities, then it holds that $p(x) = 1/2$. Considering InfoLeak, we obtain a parameterized probability distribution for the CCA presented in Section 6.1 with a parameter p_sched where p_sched denotes the probability of favourable scheduling between the sender and the receiver. As InfoLeak considers the transitions resulting in incorrectly decoding of bits as improbable, it holds that $p(y|x) = 0$ for the respective cases. The result of the capacity computation is $C = p\_sched$.

Using our empirical results for the 40%-load scenario, we estimate p_sched as the ratio between the number of favourable scheduling cases and the overall scheduling of the receiver. Thus, the capacity of the channel $C = p\_sched = 0.98$ bits per channel usage. To compare this result with the success of the attack, we calculate B denoting the ratio between the successfully transmitted bits and the number of receiver's accesses to the channel. Using our empirical data, we obtain $B = 0.78$. This confirms that InfoLeak gives an upper bound on the channel capacity and can be used to derive a feasibility metric.

Noisy vs. noiseless environment. In all the setups the validation demonstrated a correlation between the scheduling of the sender and receiver and the success ratio of the employed CCA. In the most noisy setup, the favourable scheduling cases were 905 on average out of 1200 performed decoding operations per experiment. This resulted in 534 successfully transmitted bits on average. This resulted in a strong positive correlation of 0.44 between the scheduling and the success of the attack. In the setup with 40% load the Pearson's correlation coefficient computation resulted in a very strong positive correlation of 0.8. In the noiseless setup, the mean value of the successfully transmitted bits was 1189, and the mean value of the number of favourable scheduling cases of the receiver process was 1199. In summary, in both setups, InfoLeak provided an upper bound of the capacity of the information leakage.

6.5. Discussion & Extensions

This paper defines CCs as discrete memoryless channels similar to [28], [32], and this consideration has an impact on the presented model. Hence, InfoLeak models covert-channel communication where each send, receive or other operation on the core-private cache "erases" its previous data, and previous states do not influence the capacity of the channel. As the core-private caches are relatively small, this assumption is valid as was demonstrated in Section 6.

TABLE 5. PDA P TRANSITION FUNCTION - EXTENDED MODEL.

Trans.#	Transition
6.1	$\delta(O\_i,\ r,\ s_n) = \{(U, \boldsymbol{\varepsilon})\}$
13	$\delta(O\_i,\ o,\ s) = \{(O\_i, s_n)\}$
14	$\delta(O\_i,\ o,\ s_n) = \{(U, os_n)\}$
15	$\delta(O\_i,\ s,\ s_n) = \{(O\_i, so)\}$

However, InfoLeak is extensible to also address channels with memory. We give an example considering a history of one operation, and add an additional symbol s_n (n for noise) to the stack alphabet (i.e., $\Gamma = \{Z\} \cup \{s,\ s_n,\ r,\ o\}$) for sending operation perturbed by noise. This symbol models the case when after an *S*, another process *O* causing noise has interacted with the CC. In this case we assume that the sent bit can still be decoded if *R* interacts with the channel directly after *O*. To address this case, the transitions in Table 2 simply need to be extended by the transitions shown in Table 5. This extension is only needed to model channels with memory, and not detailed here.

7. Conclusion

Being easily accessible without any special privileges, the core-private caches are often reported as a convenient CC. Although the relevance of this CC has been demonstrated, the feasibility of its usage for attacks can be influenced by varied factors e.g., the CPU scheduling.

In this paper, we proposed InfoLeak, an information leakage model focusing on the scheduling effect on the core-private cache exploitability as a CC. We comprehensively elaborated how these exploits can be impaired by the scheduling of the sender and the receiver processes, and integrated the impact into the proposed formal model. InfoLeak can assist in investigating a system regarding possible information leakage through the core-private cache, and can give an approximation on the leakage by the

scheduling information post-mortom. To validate InfoLeak, we conducted experiments with a L1 CCA in noisy and noiseless environments. Our results discern that there is correlation between the successful transmission and the favourable scheduling of the sender and receiver, and the scheduling traces can be used to detect possible CC leakage.

Cache-based CCAs and SCAs are increasing in occurrence due to their relevance in virtualized environments (e.g., the Cloud). By modeling the relationship between cache-based information leakage and the CPU scheduling, InfoLeak helps systematically ascertain efficient ways to address core-private cache covert-channel threats, and to facilitate security enhancement in shared resource systems such as the Cloud.

Acknowledgment

Research supported in part by EC H2020 MSCA-ITN NECS GA #675320 and EC H2020 CIPSEC GA #700378.

References

[1] W. M. Hu, "Lattice Scheduling and Covert Channels," in *Proc. of Symposium on Security and Privacy*, 1992, pp. 52–61.

[2] B. W. Lampson, "A Note on the Confinement Problem," *Commun. ACM*, vol. 16, no. 10, pp. 613–615, 1973.

[3] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds," in *Proc. of CCS*, 2009, pp. 199–212.

[4] Y. Zhang, A. Juels, M. Reiter, and T. Ristenpart, "Cross-VM Side Channels and Their Use to Extract Private Keys," in *Proc. of CCS*, 2012, pp. 305–316.

[5] J. Millen, "20 Years of Covert Channel Modeling and Analysis," in *Proc. of Symposium on Security and Privacy*, 1999, pp. 113–114.

[6] C. Maurice, M. Weber, M. Schwarz, L. Giner, D. Gruss, C. A. Boano, K. Römer, and S. Mangard, "Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud," in *Proc. of NDSS*, 2017.

[7] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen, and R. Schlichting, "An Exploration of L2 Cache Covert Channels in Virtualized Environments," in *Proc. of Workshop on Cloud Computing Security*, 2011, pp. 29–40.

[8] Z. Wu, Z. Xu, and H. Wang, "Whispers in the Hyper-space: High-speed Covert Channel Attacks in the Cloud," in *Proc. of USENIX Security*, 2012, pp. 159–173.

[9] V. Varadarajan, T. Ristenpart, and M. Swift, "Scheduler-based Defenses against Cross-VM Side-channels," in *Proc. of USENIX Security*, 2014, pp. 687–702.

[10] T. Kim, M. Peinado, and G. Mainar-Ruiz, "STEALTHMEM: System-level Protection Against Cache-based Side Channel Attacks in the Cloud," in *Proc. of USENIX Security*, 2012, pp. 189–204.

[11] P. Li, D. Gao, and M. Reiter, "Mitigating Access-Driven Timing Channels in Clouds using StopWatch," in *Proc. of DSN*, 2013, pp. 1–12.

[12] J. Shi, X. Song, H. Chen, and B. Zang, "Limiting Cache-Based Side-Channel in Multi-Tenant Cloud Using Dynamic Page Coloring," in *Proc. of DSN-Workshops*, 2011, pp. 194–199. [Online]. Available: http://ieeexplore.ieee.org/document/1437172/

[13] R. Martin, J. Demme, and S. Sethumadhavan, "TimeWarp: Rethinking Timekeeping and Performance Monitoring Mechanisms to Mitigate Side-channel Attacks." in *Proc. of the ISCA*, 2012, pp. 118–129.

[14] J. W. Gray, "On Analyzing the Bus-Contention Channel under Fuzzy Time," in *Proc. of Computer Security Foundations Workshop*. IEEE, 1993, pp. 3–9.

[15] C. Hunger, M. Kazdagli, A. S. Rawat, A. G. Dimakis, S. Vishwanath, and M. Tiwari, "Understanding Contention-Based Channels and Using them for Defense," in *Proc. of HPCA*, 2015, pp. 639–650.

[16] J. K. Millen, "Finite-State Noiseless Covert Channels," in *Proc. of Computer Security Foundations Workshop*, 1989, pp. 81–86.

[17] B. Köpf and D. Basin, "An Information-Theoretic Model for Adaptive Side-Channel Attacks," in *Proc. of CCS*, 2007, pp. 286–296.

[18] K. Tiri and I. Verbauwhede, "Simulation Models for Side-channel Information Leaks," in *Proc. of DAC*, 2005, pp. 228–233.

[19] T. Vateva-Gurova, N. Suri, and A. Mendelson, "The Impact of Hypervisor Scheduling on Compromising Virtualized Environments," in *Proc. of the Conference on DASC*, 2015, pp. 1910–1917.

[20] T. Vateva-Gurova, J. Luna, G. Pellegrino, and N. Suri, "Towards a Framework for Assessing the Feasibility of Side-channel Attacks in Virtualized Environments," in *Proc. of Security and Cryptography*, 2014, pp. 113–124.

[21] D. Gullasch, E. Bangerter, and S. Krenn, "Cache Games – Bringing Access-Based Cache Attacks on AES to Practice," in *Proc. of Symposium on Security and Privacy*, 2011, pp. 490–505.

[22] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-channel Attack," in *Proc. of USENIX Security*, 2014, pp. 719–732.

[23] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+Flush: A Fast and Stealthy Cache Attack," in *Proc. of DIMVA*, 2016, pp. 279–299.

[24] VMware, "Security Considerations and Disallowing Inter-Virtual Machine Transparent Page Sharing," VMware, Tech. Rep. 2080735, "Last accessed on 07.06.2018.". [Online]. Available: https://kb.vmware.com/s/article/2080735

[25] C. E. Shannon, "A Mathematical Theory of Communication," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 5, no. 1, pp. 3–55, 2001.

[26] R. A. Kemmerer, "A Practical Approach to Identifying Storage and Timing Channels: Twenty Years Later," in *Proc. of Annual Computer Security Applications Conference*, 2002, pp. 109–118.

[27] J. C. Wray, "An Analysis of Covert Timing Channels," in *Proc. of Computer Society Symposium on Research in Security and Privacy*, 1991, pp. 2–7.

[28] Z. Wang and R. B. Lee, "Capacity Estimation of Non-Synchronous Covert Channels," in *Proc. of DSN-Workshops*, 2005, pp. 170–176. [Online]. Available: http://ieeexplore.ieee.org/document/1437172/

[29] ——, "New Constructive Approach to Covert Channel Modeling and Channel Capacity Estimation," in *Proc. of ISC*, 2005, pp. 498–505.

[30] W. M. Hu, "Reducing Timing Channels with Fuzzy Time," *J. Comput. Secur.*, vol. 1, no. 3-4, pp. 233–254, 1992.

[31] J. W. Gray, "Toward a Mathematical Foundation for Information Flow Security," in *Proc. of Symposium on Security and Privacy*, 1991, pp. 21–34.

[32] I. Moskowitz and A. Miller, "Simple Timing Channels," in *Proc. of the Computer Symposium on Research in Security and Privacy*, 1994, pp. 56–64.

[33] C. Percival, "Cache Missing for Fun and Profit," in *Proc. of BSDCan*, 2005.

[34] A. Timor, A. Mendelson, Y. Birk, and N. Suri, "Using Underutilized CPU Resources to Enhance Its Reliability," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 1, pp. 94–109, Jan 2010.

[35] D. A. Osvik, A. Shamir, and E. Tromer, "Cache Attacks and Countermeasures: The Case of AES," in *Proc. of RSA Conf on Topics in Cryptology*, 2006, pp. 1–20.