Are Your Keys Protected? Time will Tell*

Yoav Ben Dov, Liron David, Moni Naor, and Elad Tzalik

Weizmann Institute of Science

Abstract. Side channel attacks, and in particular timing attacks, are a fundamental obstacle to obtaining secure implementation of algorithms and cryptographic protocols, and have been widely researched for decades. While cryptographic definitions for the security of cryptographic systems have been well established for decades, none of these accepted definitions take into account the running time information leaked from executing the system. In this work, we give the foundation of new cryptographic definitions for cryptographic systems that take into account information about their leaked running time, focusing mainly on keyed functions such as signature and encryption schemes. Specifically,
(1) We define several cryptographic properties to express the claim that the timing information does not help an adversary to extract sensitive information, e.g. the key or the queries made. We highlight the definition of *key-obliviousness*, which means that an adversary cannot tell whether it received the timing of the queries with the actual key or the timing of the same queries with a random key.
(2) We present a construction of *key-oblivious pseudorandom permutations* on a small or medium-sized domain. This construction is not "fixed-time," and at the same time is secure against any number of queries even in case the adversary knows the running time exactly. Our construction, which we call *Janus Sometimes Recurse*, is a variant of the "Sometimes Recurse" shuffle by Morris and Rogaway.
(3) We suggest a new security notion for keyed functions, called *noticeable security*, and prove that cryptographic schemes that have noticeable security remain secure even when the exact timings are leaked, provided the implementation is key-oblivious. We show that our notion applies to cryptographic signatures, private key encryption and PRPs.

1 Introduction

In any implementation of a cryptographic scheme there is a disparity between the mathematical specification of its functionality and the actual implementation in a physical device and environment. By nature, a physical implementation leaks more information than was intended and this leakage is known as a side-channel. In this work we concentrate on the running time, the side channel that is perhaps the hardest to block (and the easiest to exploit), since the time it took to perform a certain service is often known.

Consider the security of encryption and signature schemes (which we refer to as keyed cryptographic functions). After much work, definitions of the security of such schemes have been well-established for decades, and may be considered one of the crown achievements of the foundations of cryptography. But none of these accepted definitions take into account the running time information leaked from executing the system.

In this work, we give a foundation for defining the security of cryptographic systems that take into account that their running time is *leaked*, focusing mainly on keyed functions such as signature and encryption schemes. More specifically, we suggest several cryptographic definitions for scenarios where the leaked information does not help an adversary to expose sensitive information, e.g. the key or the queries made. The most interesting definition we propose is *key-oblivious*. For this notion we prove that for cryptographic schemes such as digital signatures, private key encryption and pseudorandom permutations (PRPs), if their implementation satisfies key-obliviousness, then they preserve their security even when the *exact* timing is leaked. Finally, we construct a PRP called "Janus Sometimes Recurse (JSR)" that is not fixed-time, yet provably secure against timing attacks (key-oblivious and other properties).

* Research supported in part by grants from the Israel Science Foundation (no.2686/20), by the Simons Foundation Collaboration on the Theory of Algorithmic Fairness and by the Israeli Council for Higher Education (CHE) via the Weizmann Data Science Research Center.

For a motivating example of the JSR construction, consider the following question, taken more or less verbatim from Stack Overflow[1]:"*I am looking to enumerate a random permutation of the numbers* $1 \dots N$ *in fixed space. I cannot store all numbers in a list,* N *can be very large, more than available memory. I still want to be able to walk through such a permutation of numbers one at a time, visiting each number exactly once.*"

As we shall see, there are good cryptographically based solutions to the question, but they are susceptible when the creator of the permutation leaks how long it took to choose the next value. In light of the work on JSR in Section 4.1 we have a good solution that is immune even when the timing information is leaked. See Appendix A.

1.1 A Brief History of Timing Attacks

There is a significant body of research about side-channel attacks and more specifically timing attacks, and how to exploit them in order to break cryptographic protocols. An early work by Lipton and Naughton [24] showed a way to exploit timing information to compromise the performance of dictionaries that employ universal hash functions.

Kocher [23] showed how the running time of certain implementations of RSA and Diffie Hellman schemes leaks information that can be used to recover the prime factors or find the discrete log, hence breaking the systems. In more detail, Kocher showed that the running time of some implementations depends on the exponent chosen in the protocol, and by carefully timing the running time on multiple outputs one can extract information about the exponent, which can then be used to break the security of the protocol. Kocher's work brought widespread attention to the crucial importance of implementations of cryptographic protocols, such as public-key encryption and signatures, and led to a considerable body of research on vulnerabilities to side-channel attacks and spurring studies and advancements aimed at strengthening their security against such attacks.

Brumley and Boneh [11] showed that timing attacks are practical on large systems and over the web. In large systems, the response time suffers significantly from noise coming from latency, multi-threading, communication bottlenecks and more. Brumley and Boneh showed that even under these conditions it is possible to retrieve the private key with timing attacks on OpenSSL servers. More specifically, they showed how to reconstruct the prime factors used in an RSA protocol by making about one million queries and carefully inspecting the response times.

One of the most efficient lattice-based digital signature schemes is BLISS, suggested by Ducas, Durmus, Lepoint and Lyubashevsky [14]. This scheme uses a bimodal Gaussian sampler and was shown to be vulnerable to timing attacks, and, in particular, the sampling component that is not independent of the secret-key [15,10], as well as other attacks.

All of these examples are but a drop in the ocean of the vast and rich research area of side-channel attacks. New and more sophisticated and subtle attacks and vulnerabilities are found every once in a while, a solution is suggested and implemented, and then another attack is found, in what feels like a never-ending game of cat and mouse. For a more thorough (but still far from full) overview of the history and background of side-channel attacks and timing attacks in particular see Crosby, Wallach and Riedi [13] and Biswas, Ghosal and Nagaraja [8].

1.2 Prevention Techniques

The main approach to prevent timing attacks is to use *fixed-time algorithms*, *often called in the literature "constant time algorithms,"* meaning algorithms that run the same *amount of* time on all inputs.

There are two main drawbacks to this solution. First, in order for the algorithm to run in fixed-time on all the inputs, we need to know the worst-case running time, a task that is often challenging on its own. The second one is that even if we do know the running time, in many cases there is a very large gap between *best case and worst case running time*, or even *average case and worst case running time*, and by making

[1] `https://stackoverflow.com/questions/10054732/create-a-random-permutation-of-1-n-in-constant-space`.

the algorithm run in the worst case time on all inputs, we create huge overheads. It is also worth mentioning that the second caveat can make many protocols and algorithms impractical and not usable when efficiency is critical.

In addition, the survey in Section 1.1 demonstrates that the task of making an algorithm run in fixed-time is more subtle and challenging than meets the eye. Timing information might leak from the response times of the server, from I/O calls, from reading RAM memory or cache memory, and many more possibilities. For the implemented algorithm to be truly and fully fixed-time one must make sure to make everything fixed-time, which is often very challenging and goes against hardware and software optimizations.

A common technique to thwart timing attacks in the public-key context is *"blinding,"* first suggested by Chaum [12] in the context of signatures, where a value v is mapped into a random-looking one u prior to the encryption or signature, in a manner that allows retrieving the desired signature or encryption from the encryption or signature on u. Kocher [23] suggested using blinding to make RSA implementations secure against timing attacks. The blinding works by multiplying the input x by a fresh random element r of the group ${\mathbb{Z}_N}^*$, i.e. a random element which is co-prime to N. To decode, multiplication by the group inverse r^{-1} is done at the end of the computation. Note that simply using the same r for many inputs will not work, as the attack suggested by Kocher can recover r over time, and even recover the exponent without knowing r. Hence, fresh r needs to be chosen in each round. This example goes to show that using blinding as a technique to protect against timing attacks is often a subtle task, and that if implemented naively or incorrectly can lead to a false sense of security.

A general approach to preventing leakage is to employ techniques from secure multi-party computation, and split the input into various parts where leaking *almost* all of the parts does not reveal the actual values. It was first suggested in Ishai, Sahai and Waters [20] for thwarting probing attacks (see Kalai and Reyzin [21] for a survey). This can be thought of as the "moral equivalent" of blinding for a general function. However, in the case of timing, at the very least the *sum* of their running times is leaked (since they are all executed on the same machine, what is leaked in the *total* time of all the emulated processors) and this is a function of *all* parties. Hence this does not solve the problem, unless an argument is made that the sum of all the execution times is not meaningful.

Another set of techniques is known as the bounded retrieval model (see [2] for a survey). In this model, the adversary learns some arbitrary function of the secret key that is shorter than the key. An alternative is the noisy leakage model, where the leakage is not of bounded length but it is guaranteed that the secret key is still unpredictable given the leaked value [31]. But this is not the case in our setting, with a repeatedly used keyed function: the adversary learns the timing of the keyed function on *many* inputs. Altogether this leaked information may be much greater than the key size. There is also the continual-leakage model, which is more appropriate for this case. The work of Goldwasser and Rothblum [18] considered leakage with an unbounded number of executions, in the presence of an adversary who observes partial information on the internal state of the computation during the executions. They showed that it is possible to obtain secure computation in the sense that the adversary learns only input-output behavior if the leakage in any round is bounded (following the 'only computation leaks information' maxim of Micali and Reyzin [28]). However, this is not a silver bullet for timing attacks, as in this work the first step is to turn the program to be computed into a circuit, i.e. into a fixed-time computation - and this carries over the various downsides of this approach, for instance, that typical case becomes the worst case. Nevertheless, such an approach may be useful for various critical sections when one wants to get fixed time. *Our goal in this work is to relax fixed time and allow information related to the key to leak, but specify what it means to say that it is not harmful.*

Extending the Notion of Constant Time: There have been a number of proposals to extend the notion of constant time implementations in order to argue that no meaningful information is leaked from the timing. For instance, Benegas et al. [4] talk about the distribution of the running time being the same for any key and any input. Similarly, Almeida et al. [1] define a program to be secure if all equivalent programs in terms of inputs and outputs are indistinguishable given the leaked information, i.e. it "means that any two executions whose input and output values differ only with respect to secret information must leak exactly the same observation." These extensions are not flexible enough to talk about protecting keyed

cryptographic functions, since the protection there is computational, and the inputs and outputs are not going to be identical. For instance consider the case of signature schemes.

Note that we use the term 'fixed-time' since in the literature constant time sometimes does not refer to operations that take the same amount of time no matter what the input is.

1.3 Comparing Our Work With the Existing Ones

At this point, the reader may be wondering whether enough theoretical work was already done in the area of leakage and there is not much to add. The novel aspect of our work is proposing criteria for arguing that the leakage is benign, that is, the presence of this benign leakage, although not being fixed, does not compromise the original guaranteed security in many cases. Maintaining this criteria is, therefore, sufficient to argue security even with the presence of the leakage in many cases. An illustrative example is the famous GGM construction of pseudorandom functions (PRF) F from length-doubling pseudorandom generators (PRG) G. *What properties should we require of the PRG G in order to argue that F is secure?* Recall that the construction is defined by imagining a full binary tree of depth n where each node gets an n bit label. The root is labeled with the key k and each parent induces a labeling of its two children by applying G to its label; the left half of the result becomes the label of the left child and the right half the label of the right child. Clearly, requiring that G be fixed time and making the rest of operations (deciding whether to branch left or right based on the bit) fixed-time is sufficient. But can we get a weaker requirement from G and how to express it? What happens when the construction is not applied a fixed number of times, but one that can vary with the input? Could such a construction be secure?

In this work we define a formal condition that is sufficient to argue security in the presence of leakage in many cases. We call this condition "key-oblivious." Namely, in order to prove that a construction is secure in the presence of leakage, one only needs to prove that the construction is key-oblivious. The key-obliviousness then implies security in the presence of leakage. We argue that this notion is easier to reason about than directly proving that the leakage does not hurt security.

A possible comparison is to the definition of security of encryption. The "moral equivalent" of this condition is the notion of indistinguishability of encryptions, which is, generally speaking, easier to prove than semantic security. But we know that the two notions are equivalent in that context.

Note that the notion of key-oblivious is relevant to any type of leakage, not necessarily timing, but in case of time we have various properties that make it particularly useful, e.g, the leakage of applying a function f and then g is, under reasonable assumptions, the sum of the two leakages.

1.4 Our Contributions and Technical Overview

Our goal in this work is to investigate the landscape of algorithms and systems that can be implemented in a manner resistant to timing attacks, but we wish to expand the 'Procrustean bed' of fixed-time algorithms. We provide foundational treatment to the subject as well as many algorithms and separation results.

We propose several criteria for expressing the property that the timing information of an implementation of an algorithm does not expose sensitive information in the context of keyed functions. The most interesting one is *key-oblivious* (Definition. 1), which means that a polynomially bounded adversary cannot tell whether it received the timing of the actual key or of a random unrelated key. Namely, suppose that $\mathcal{F}_k$ is a keyed function with a key k and $\mathcal{T}(\mathcal{F}_k(q))$ is the time takes to execute $\mathcal{F}_k$ on the query q, then the key oblivious definition means that a PPT adversary cannot distinguish the following two cases: whether the time it gets is the real running time on the actual key k, or whether the running time is on an unrelated key k':

Definition 1 We say a keyed function $\mathcal{F}$ is **key-oblivious** secure against timing attacks if any probabilistic polynomial-time (PPT) adversary Adv has a winning probability at most $\frac{1}{2} + \mathsf{negl}(n)$ in the following game:

1. Two keys are sampled k_0, k_1.
2. A random bit $b \in \{0, 1\}$ is sampled.
3. The adversary Adv makes $\ell = poly(n)$ adaptive queries $q_1, \ldots, q_\ell$ to $\mathcal{F}_{k_0}$, and gets $\mathcal{F}_{k_0}(q_1), \ldots, \mathcal{F}_{k_0}(q_\ell)$, as well as $\mathcal{T}(\mathcal{F}_{k_b}(q_1)), \ldots, \mathcal{T}(\mathcal{F}_{k_b}(q_\ell))$.

4. The adversary outputs b', the guess of b, and wins if $b' = b$.

How useful is this criterion? What does it imply? The notion of key obliviousness is most useful in cases where the period where the adversary has access to the timing information is separated from when it actually attacks; for instance, in the case of signatures schemes, where the adversary may know how long it takes to produce a signature on a message, but where the adversary does not have access to the timing information of the signing of the actual message it wants to forge. We then prove that if we have signature scheme $\mathcal{F}_k$ that is existentially unforgeable secure against an adaptive chosen message attack and the signature function $\mathcal{F}_k$ is key-oblivious, then even if the adversary in the forgery game has access to the running time it takes to generate the signatures, then this adversary will not manage to forge a valid-looking signature on any message it was not given a signature explicitly.

As mentioned above, the key-oblivious criteria is most relevant when the attack occurs after timing information is not available anymore, e.g. as in the case of signature schemes. But there are scenarios where this is not the case and the adversary does get timing information during a "challenge phase." Consider, for instance, the case of encryption, where the final goal of the adversary is to distinguish between the encryption of two messages. The game has a "challenge phase" in which the adversary sends two messages and receives an encryption of one of them and its goal is to guess which one it is. The encryption may not be time-secure, even though the encryption implementation is key-oblivious.

To see this, consider the following example: suppose that the running time depends only on the least significant bit (lsb) of the message and does not depend on the key or other bits of the message. Then given two messages with different lsbs, the adversary who gets the running time of the actual message that was chosen, can easily distinguish whether the encryption was of one message or the other.

A case where this may be significant is in voting machines where votes are encrypted and then shuffled. If the timing of an encryption of a particular vote is known, then if the encryption is not *query-oblivious* in the above sense then this yields information about the actual vote.

To guarantee time-security also in cryptographic games that have a "challenge phase" (as in the indistinguishability game) we propose another security criteria called *query-oblivious* (Definition 3) whose aim is to capture the property that the time to evaluate a query does not leak information about the query itself. We then prove specifically for indistinguishability of encryptions (Theorem 16) that if the implementation is query-oblivious, then it is time-secure.

A fundamental issue concerning any new security definition is what happens when a primitive satisfying it is part of a larger structure and whether the new criterion is preserved under different constructions. To this end, we investigate different constructions and explore whether they preserve key-obliviousness and whether being query-oblivious as well is necessary for them to preserve key-obliviousness. We focus on the following constructions:

- The famed Goldreich-Goldwasser-Micali (GGM) construction: we show that if G is a PRG implemented in a key-oblivious manner, then applying GGM with G yields a PRF that is key oblivious.
- The cycle walking technique for format-preserving encryption: we show that if the permutation π is key oblivious, then the result π' is key oblivious.
- Domain extensions of PRFs: we show that even if the underlying PRFs are key-oblivious, then the classical results do not necessarily imply that the result is key-oblivious. But we show that the *cascading construction of PRF extension preserves key obliviousness.*

Main Application: In Section 4 we turn our attention to pseudorandom permutations (PRP) on small domains (related to format-preserving encryption). The most efficient construction for small domain PRP is the "Sometimes Recurse" (SR) shuffle by Morris and Rogaway [29], which runs in *expected* time of $O(\log N)$ and is secure even when the adversary queries the whole domain (a more detailed exposition appears in Section 4). The downside of SR is that its running time is fully determined by the number of leading 1's in the output. This makes the SR construction not secure against timing attacks, namely, SR is neither key-oblivious (Claim 9) nor query-oblivious (Claim 10).

We suggest a new construction of a PRP on small domains which we call the *Janus Sometimes Recurse* (JSR) *that is not constant time, yet provably secure against timing attacks. Our construction is faster than all previously known constructions that are secure against timing attacks.* Specifically we prove that: (1) JSR is key-oblivious (Claim 11), i.e., a PPT adversary cannot distinguish between the key that was used and a random key even when the adversary gets the exact running time of the PRP; and (2) JSR is also *query-oblivious* (Claim 13), i.e., a PPT adversary cannot infer from the computation time of the PRP on a query q what q is.

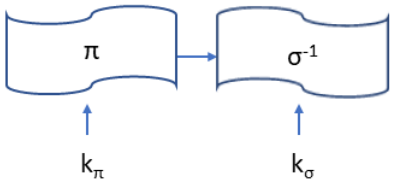

Fig. 1. JSR construction on two PRPs

Generally speaking, JSR takes two independent (i.e. with two independent keys) copies of SR on the same domain $[N]$, where the permutations are denoted by π and σ and the keys by k^π and k^σ, and composes π with σ^{-1}, see Figure 1. This is similar to the approach that Maurer and Pietrzak [26] used to move from non-adaptive to adaptive PRPs. The term "Janus" in the name of our construction "Janus Sometimes Recurse" (JSR) comes from the Roman god who was depicted as having two faces, since both the directions (encryption and decryption) are forward-looking.

The intuition for this construction is that while the running time of the forward direction leaks information about the output, the running time of the inverse is *determined by the input*, and so by composing the two we get that the running time of the algorithm both in the forward direction and in the inverse, is determined by *the inner value* which is *almost* independent of the input and output since π, σ are PRPs.

In Appendix A we show an application of the above claims to the low memory generation of a random permutation of the numbers in a given range $1 \ldots N$.

Main Security Claim: To formally define the criteria in general, we consider a *cryptographic game* (Definition 6) for a keyed function, which captures the security of many primitives including indistinguishability of encryptions, digital signatures, and pseudo-random permutations (we denote for function $\mathcal{F}$ the security game with $G_{\mathcal{F}}$). A game $G_{\mathcal{F}}$ has *noticeable security* (Definition 6), if it is defined between an adversary and a principal, and determining who wins the game can be done without direct access to the key, but simply based on the queries and the state of the principal. We show that digital signatures, pseudo-random permutations, and encryption (but for indistinguishability of encryption see caveat below and Section 5.4) have games with noticeable security.

The main result (Theorem 1) shows that: For *a keyed function $\mathcal{F}_k$ that is secure w.r.t. a game $G_{\mathcal{F}}$ that has "noticeable security," if the implementation of $\mathcal{F}_k$ is key-oblivious then $\mathcal{F}_k$ is time-secure, that is, $\mathcal{F}_k$ is secure w.r.t. $G_{\mathcal{F}}$ even when the exact running timing of executing the oracle on the queries is leaked to the adversary.*

2 Keyed Functions Secure Against Timing Attacks

Our goal in this section is to present the definitions of timing-resistant keyed functions. In Section 2.1 we provide three criteria and in Section 2.2 we look at a specific keyed function, the pseudo-random function (PRF), and we show that if the implementation of PRF is key-oblivious then it is time-secure (later on, in Section 5 we generalize this to any function with noticeable security).

2.1 Definitions of Timing Resistance of Keyed Functions

We now aim to formalize security against timing attacks for keyed cryptographic functions. Since the security of keyed cryptographic functions is usually measured by the success of a PPT adversary in some game, we would like the security notion to provide the keyed function security against such adversaries.

Let $\mathcal{F}$ be a keyed function, with key space $\{\mathcal{K}_n\}_n$, where keys are sampled from $\mathcal{K}_n$ have length $poly(n)$. Denote by $\mathcal{F}_k$ the keyed function with a chosen key k. To ease the notation we will use $k \sim \mathcal{K}$ to denote sampling k from $\mathcal{K}_n$ when n is understood from context.

We denote by q a query to the function. We do not state what type of query it is, since different types of functions will have different queries. For example, if $\mathcal{F}$ is an encryption scheme, then it makes sense to allow encryption queries as well as decryption queries. Denote by $\mathcal{F}_k(q)$ the answer to the query and by $\mathcal{T}(\mathcal{F}_k(q))$ *the running time* it took for the answer to return.

Assumption on Running Time Running time is implementation dependent, hence we stress that $\mathcal{T}(\mathcal{F}_k(q))$ depends on $\mathcal{F}, q, k$ as well as the implementation of $\mathcal{F}$ in the computational model. In many cases, once a key is fixed, the running time on a query will be deterministic, but there are cases in which the running time might be a distribution even with the same key and query. We therefore think of $\mathcal{T}$ as a distribution (which may be a distribution supported on one element).
The crucial assumption which is *running time specific* that we assume is *linearity of composition*, meaning that $\mathcal{T}(\mathcal{F} \circ \mathcal{G}(x)) = \mathcal{T}(\mathcal{F}(\mathcal{G}(x))) + \mathcal{T}(\mathcal{G}(x))$ where by $\mathcal{F} \circ \mathcal{G}$ means running $\mathcal{G}$ on x, and sequentially $\mathcal{F}$ on $G(x)$. This assumption is not used in this section, as well as Section 5, since in these sections we study a single function. On the other hand linearity of composition is a key for designing complex cryptographic primitives from basic ones, e.g. in the JSR construction appearing in Section 4.
We, therefore, require that for some of the statements in the paper *at certain points of the computation*, the model is inherently sequential and that many optimizations incorporated by modern computers to speed running time (e.g. pipeline, multi-processing, branch prediction, etc.) are not allowed at those points (hence the linearity assumption). As is well known, such optimizations can be exploited, with Spectre being one of the notable examples.

We use the notation negl for any function $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}^+$ satisfying that for every positive polynomial $p(\cdot)$ there is an N such that for all integers $n > N$ it holds that $\mathsf{negl}(n) < \frac{1}{p(n)}$. Such functions are called *negligible*. We will also call a random variable with distribution $Bernoulli(\frac{1}{2})$ a random bit.

We state three definitions of security of keyed cryptographic function. The first two definitions concern securing the key from a timing attack, while the third is designed to secure the result of queries.

Definition 1. *We say a keyed function $\mathcal{F}$ is* **key-oblivious** *secure against timing attacks if any probabilistic polynomial-time (PPT) adversary* Adv *has a winning probability at most $\frac{1}{2} + \mathsf{negl}(n)$ in the following game:*

1. *Two keys are sampled $k_0, k_1 \sim \mathcal{K}$.*
2. *A random bit $b \in \{0,1\}$ is sampled.*
3. *The adversary* Adv *makes $\ell = poly(n)$ adaptive queries $q_1, \ldots, q_\ell$ to $\mathcal{F}_{k_0}$, and gets $\mathcal{F}_{k_0}(q_1), \ldots, \mathcal{F}_{k_0}(q_\ell)$, as well as $\mathcal{T}(\mathcal{F}_{k_b}(q_1)), \ldots, \mathcal{T}(\mathcal{F}_{k_b}(q_\ell))$.*
4. *The adversary outputs b', the guess of b, and wins if $b' = b$.*

Definition 1 means that the joint distribution of running times on a polynomial number of queries for two keys $\mathcal{T}(\mathcal{F}_{k_b}(q_1)), \ldots, \mathcal{T}(\mathcal{F}_{k_b}(q_\ell))$ are indistinguishable by a PPT adversary even when it sees the results of the query on a specific key. We now strengthen Definition 1:

Definition 2. *We say a keyed function $\mathcal{F}$ is* **key-switch** *secure against timing attacks if any PPT adversary* Adv *has a winning probability at most $\frac{1}{2} + \mathsf{negl}(n)$ in the following game:*

1. *Two keys are sampled $k_0, k_1 \sim \mathcal{K}$.*
2. *The adversary* Adv *makes $\ell = poly(n)$ many queries $q_1, \ldots, q_\ell$ and gets $\mathcal{F}_{k_0}(q_1), \ldots, \mathcal{F}_{k_0}(q_\ell)$ as well as $\mathcal{T}(\mathcal{F}_{k_0}(q_1)), \ldots, \mathcal{T}(\mathcal{F}_{k_0}(q_\ell))$.*

3. *A random bit* $b \in \{0,1\}$ *is sampled.*
4. *The adversary* Adv *makes another* $\ell' = poly(n)$ *many queries* $p_1, \ldots, p_{\ell'}$ *and gets* $\mathcal{F}_{k_0}(p_1), \ldots, \mathcal{F}_{k_0}(p_{\ell'})$*, as well as* $\mathcal{T}(\mathcal{F}_{k_b}(p_1)), \ldots, \mathcal{T}(\mathcal{F}_{k_b}(p_{\ell'}))$.
5. *The adversary outputs* b'*, the guess of* b*, and wins if* $b' = b$.

Notice that if we skip step 2 we get back to Definition 1. The difference here is that the adversary gets the running time of the function with the same key as the answers, until a bit is chosen, and only then the adversary does not know if the timing comes from the same key or a different key. This essentially means that the distribution of the running times $\mathcal{T}(\mathcal{F}_{k_b}(p_1)), \ldots, \mathcal{T}(\mathcal{F}_{k_b}(p_{\ell'}))$ conditioning on the answers to the queries and the running time on the original key are indistinguishable by a PPT adversary.

This definition could be useful to prevent denial-of-service attacks: if the adversary can find expensive (time-wise) queries, then it can bunch expensive queries together and ask them so as to overload the system. If, in addition, the system has the property that a priory it is not clear how long a query would take, then it is not possible to find expensive queries (since then it would be possible to figure out whether a key-switch occurred or not).

The third definition is of *query-obliviousness* and involves only a single key and aims to express that the actual queries made are secure from a timing attack. This is desirable, for instance, in voting systems. Consider a voting system that uses a keyed function (e.g. a PRP) as a subroutine to the actual vote cast. The sensitive information that needs to be protected is the votes themselves and not necessarily the key.

Definition 3. *We say a keyed function* $\mathcal{F}$ *is* **query-oblivious** *secure against timing attacks if any PPT adversary* Adv *has a winning probability at most* $\frac{1}{2} + \mathsf{negl}(n)$ *in the following game:*

1. *A single key* $k \sim \mathcal{K}$ *is sampled.*
2. *The adversary* Adv *makes* $\ell = poly(n)$ *adaptive queries* $q_1, \ldots, q_\ell$ *and gets* $\mathcal{F}_k(q_1), \ldots, \mathcal{F}_k(q_\ell)$ *and* $\mathcal{T}(\mathcal{F}_k(q_1)), \ldots, \mathcal{T}(\mathcal{F}_k(q_\ell))$ *of all the queries.*
3. *The adversary* Adv *chooses two new distinct queries* $q'_0 \neq q'_1$ *such that* $q'_0, q'_1 \notin \{q_1, \ldots, q_\ell\}$.
4. *A bit* $b \in \{0,1\}$ *is chosen at random.*
5. *The adversary* Adv *gets* $\mathcal{T}(\mathcal{F}_k(q'_b))$.
6. *The adversary outputs* b'*, the guess of* b*, and wins if* $b' = b$.

In the game above, the adversary makes queries and gets their running time. Then the adversary chooses two different queries and gets the running time of one of them. The challenge is to decide what is the query whose running time was returned. There are two (non-mutually exclusive) variations on this:

Weakly vs. Strongly: If the function is also secure for general queries q'_0, q'_1 that are *not necessarily new*, then we say that function is *strongly query-oblivious*, while the definition above is *weakly query-oblivious*.

With vs. Without Results We define a variant of query-oblivious, we call **query-with-results-oblivious** which is the same as the above query-oblivious game but with one change: in step (5) the adversary Adv gets both $\mathcal{F}_k(q'_b)$ and $\mathcal{T}(\mathcal{F}_k(q'_b))$ rather than only $\mathcal{T}(\mathcal{F}_k(q'_b))$. As we shall see, this query-with-result-oblivious will be useful, for example, for time-security of indistinguishability of encryption, while the original query-oblivious definition will be useful, for example, for domain extension.

Remark 1. The definitions above are not equivalent to one another, see Appendix B for formal arguments.

2.2 Example: Pseudorandom Functions

We start by demonstrating the definition of key-oblivious for a specific primitive, that of pseudorandom function. We prove that if the implementation of a PRF is key-oblivious then the PRF is time-secure, that is, the PRF is secure w.r.t. the cryptographic game in which the adversary has access not only to the PRF oracles but also to the time it takes the oracle to execute. The general security for all primitives with so-called noticeable security is discussed in Section 5.

Recall the definition of PRF:

Definition 4. *(Pseudo-random Function) Let $\mathcal{F} : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^n$ be a keyed function in which the first input is called the key. We denote by $\mathcal{F}_k(x) := \mathcal{F}(k,x)$ the keyed function with the key $k \in \{0,1\}^m$. Let $\mathsf{Adv}_{\mathsf{PRF}}$ be a probabilistic polynomial-time adversary in the following game:*

1. *A key $k \in \{0,1\}^m$ is sampled.*
2. *A random bit b is sampled.*
3. *$\mathsf{Adv}_{\mathsf{PRF}}$ chooses adaptively $\ell = poly(n)$ queries $q_1, \ldots, q_\ell$. If $b = 0$ then it receives the values of $\mathcal{F}_k$ on each query*
$$y_1 = \mathcal{F}_k(q_1), \ldots, y_\ell = \mathcal{F}_k(q_\ell),$$
If $b = 1$ then it receives the values of a random function f on each query
$$y_1 = f(q_1), \ldots, y_\ell = f(q_\ell),$$
4. *Adversary $\mathsf{Adv}_{\mathsf{PRF}}$ guesses b' and wins if $b' = b$.*

The function $\mathcal{F}_k$ is a pseudo-random function if for all PPT adversaries $\mathsf{Adv}_{\mathsf{PRF}}$ as above there is a negligible function negl such that, for all n,
$$\Pr[b = b'] \leq 1/2 + \mathsf{negl}(n).$$

To define a time-secure implementation of a PRF we let the distinguisher receive the timing information as well:

Definition 5. *(Time-secure PRF) An implementation of a PRF is time-secure if the winning probability in the cryptographic game in Definition 4 remains $0.5 + \mathsf{negl}(n)$ even when the adversary gets, not only the answers for the oracles queries (namely either $\mathcal{F}_k(q_i)$ or $f(q_i)$), but also the time it takes for the oracle $\mathcal{F}_k$ to execute them, namely $\mathcal{T}(\mathcal{F}_k(q_i))$. Note that in case the queries are answered by a random function, the only output related to k is the running time $\mathcal{T}(\mathcal{F}_k(q_i))$.*

We prove that if the implementation of PRF is key-oblivious, then it is also time-secure.

Theorem 1. *Let $\mathcal{F}_k$ be a PRF. If the implementation of $\mathcal{F}_k$ is key-oblivious, then the implementation of $\mathcal{F}_k$ is time-secure.*

Proof. Suppose that there exists an adversary $\mathsf{Adv}_{\mathsf{TS}}$ attacking the time-security of a PRF that is presumed to be key-oblivious. Consider four games, similar to the original ones (with the exception that sometimes a new key k' is chosen) and the associated probabilities that the adversary outputs '1':

p_1: The adversary $\mathsf{Adv}_{\mathsf{TS}}$ is given the value $\mathcal{F}_k(q_i)$ on query q_i and the timing of $\mathcal{T}(\mathcal{F}_k(q_i))$.
p_2: The adversary $\mathsf{Adv}_{\mathsf{TS}}$ is given the value of the random function at q_i and the timing $\mathcal{T}(\mathcal{F}_k(q_i))$.
p_3: The adversary $\mathsf{Adv}_{\mathsf{TS}}$ is given the value $\mathcal{F}_k(q_i)$ and the timing $\mathcal{T}(\mathcal{F}_{k'}(q_i))$ for another key k' chosen at random.
p_4: The adversary $\mathsf{Adv}_{\mathsf{TS}}$ is given the value of the random function at q_i and the timing $\mathcal{T}(\mathcal{F}_{k'}(q_i))$ of the other key k'.

We know that $p_2 = p_4$, since from $\mathsf{Adv}_{\mathsf{TS}}$'s point of view there is no difference between k and k'. If there is a non-negligible difference between p_1 and p_3, then we can use it to break the key-oblivious assumption of $\mathcal{F}$. If there is a non-negligible difference between $p_2 = p_4$ and p_3, then we can use it to mount an attack of the pseudorandomness of $\mathcal{F}$ without timing information: choose a key k' and whenever a query q_i arrives add the timing information of $\mathcal{T}(\mathcal{F}_{k'}(q_i))$. □

On the other hand, we can show

Theorem 2. *Let $\mathcal{F}_k$ be a PRF. If $\mathcal{F}_k$ implementation is* not *key-oblivious, then it is not time-secure.*

Proof. Similarly to the proof above, suppose that there exists an adversary $\mathsf{Adv}_{\mathsf{KO}}$ attacking the key-obliviousness of a PRF that is presumed to be time-secure. Consider four games, similar to the original ones and the associated probabilities that the adversary outputs '1':

p_1: The adversary $\mathsf{Adv}_{\mathsf{KO}}$ is given the value $\mathcal{F}_k(q_i)$ on query q_i and the timing of $\mathcal{T}(\mathcal{F}_k(q_i))$.
p_2: The adversary $\mathsf{Adv}_{\mathsf{KO}}$ is given the value $\mathcal{F}_k(q_i)$ on query q_i and the timing of $\mathcal{T}(\mathcal{F}_{k'}(q_i))$ for another key k' chosen at random.
p_3: The adversary $\mathsf{Adv}_{\mathsf{KO}}$ is given the value of the random function at q_i and the timing $\mathcal{T}(\mathcal{F}_k(q_i))$.
p_4: The adversary $\mathsf{Adv}_{\mathsf{KO}}$ is given the value of the random function at q_i and the timing $\mathcal{T}(\mathcal{F}_{k'}(q_i))$ of the other key k'.

We know that $p_3 = p_4$, since from $\mathsf{Adv}_{\mathsf{KO}}$'s point of view there is no difference between k and k'. If there is a non-negligible difference between p_1 and p_3, then we can use it to break the time-security assumption of $\mathcal{F}$. If there is a non-negligible difference between $p_3 = p_4$ and p_2, then we can use it to mount an attack of the pseudorandomness of $\mathcal{F}$ without timing information: choose a key k' and whenever a query q_i arrives add the timing information of $\mathcal{T}(\mathcal{F}_{k'}(q_i))$. □

Note that what we proved for pseudorandom functions (PRFs) is true also for pseudorandom permutations (PRPs) and key-oblivious implies that they are secure even with timing information. As we mentioned, we generalize this to many primitives such a signatures schemes in Section 5.

3 Constructions Preserving Key-Obliviousness

A natural question about the notion of key-oblivious is whether the key-oblivious is preserved when applying it to several functions. In this section we investigate several well known constructions and check whether the key-obliviousness is preserved under these constructions, given that the underlying building blocks are key-oblivious. We start with the basic constructions of composition and concatenation. We then consider the fundamental cryptographic constructions:

1. The GGM construction of pseudorandom functions, where we show that if the basic building block, the PRG G, is key oblivious, then the result is key oblivious (Claim 3). However it is not necessarily query-oblivious (Claim 4).
2. We consider the Cycle walking technique for constructing format-preserving encryption, which is not fixed time by nature, yet we show that if the underlying permutation π is key oblivious, then the result is key oblivious.
3. Finally we consider various *domain extension techniques* and show that while some of them do not preserve key-obliviousness (e.g. using the Levin trick) it is possible to get key-obliviousness using either an additional primitive such as UOWHF or using the cascading consturction.

Composition and concatenation: A basic issue when considering security definition is how they interact as part of a larger system. The good news is that wrt concatenation key-obliviousness is preserved: Suppose that we have two keyed functions f and g and suppose that their keys are independent. Then the natural implementation of producing $f(x) \circ g(x)$, first compute $f(x)$ and then $g(x)$ is also key oblivious. Also suppose that at each step either f or g are called, then the whole process is still key-oblivious. Furthermore, let h be any function that is implemented in constant time (e.g. addition or Xor). Then the natural implementation of $h(f(x), g(x))$ where f and g are key-oblivious and are computed in a sequential manner and where h is fixed time is itself key oblivious.

On the other hand, as we shall see, for composition the case is different: even if f and g are key-oblivious it is not necessarily true that $f(g(x))$ is key oblivious! This is shown in Claim 6.

3.1 The GGM Construction of PRFs

Consider the Goldreich-Goldwasser-Micali (GGM) construction of pseudorandom functions (PRF) from pseudorandom generators [16]. The construction starts with a pseudorandom generator $G\colon \{0,1\}^n \mapsto \{0,1\}^{2n}$ and the PRF $\mathcal{F}_k\colon \{0,1\}^n \mapsto \{0,1\}^n$ with key $k \in \{0,1\}^n$ is defined by imagining a full binary tree of depth n where each node gets an n bit label. The root is labeled with the key k and each parent induces a labeling

of its two children by applying G to its label; the left half of the result becomes the label of the left child and the right half the label of the right child. The value of $\mathcal{F}_k(x)$ for $x \in \{0,1\}^n$ is the label of the leaf at the end of the path defined by x.

Suppose that G is implemented in a key oblivious manner, meaning in this case, simply that given $G(k)$ for a uniform $k \in \{0,1\}^n$ and $\mathcal{T}(G(k))$ or $\mathcal{T}(G(k'))$ for a uniform $k' \in \{0,1\}^n$, it is hard for a poly-time adversary to distinguish between the two cases. Now consider the straightforward implementation of the GGM PRF $\mathcal{F}_k(x)$ from G, which consists of n applications of G given k and x (developing the required labels). Assume that taking the left or right half of the output of G once it is computed is fixed-time and that each application of G starts from scratch. Is the result key oblivious? Is it query-oblivious?

Claim 3 *The key-obliviousness of G together with the requirement that it is a PRG imply that the GGM construction is key-oblivious.*

Proof. One possible way to prove the claim is to follow the same lines as the classical proof of pseudo-randomness of the GGM construction. This proof is based on a hybrid argument. If it is possible to distinguish between the construction and a truly random function using m queries, then there is a sequence of $m' \leq m \cdot n$ distributions, the first being the pseudorandom one, as described above and the last one being the truly random one[2]. An alternative approach is to use induction on the depth on the tree. For the base case $n = 1$, the property that G is assumed to have is sufficient to guarantee key obliviousness. To increase the number of levels, we will think of the two branches from the root as two independent functions. In this case, the whole process should still be key oblivious, as the discussion at the beginning of Section 3 shows. If the actual implementation is no key-oblivious, then again, we have an attack of the key obliviousness of G.

Claim 4 *The GGM construction is* not *query-oblivious, at least not if G is not fixed-time computable. This is true even if we consider weakly query-without-result-oblivious.*

Proof. Note that the time to compute $\mathcal{F}_k(x)$ is the sum of n applications of G on random-looking inputs. Furthermore, the timings of $\mathcal{F}_k(x)$ and $\mathcal{F}_k(x')$ for x and x' that differ only in the last bit are closely correlated (since the sums are over the same summands, except the last one), compared to the timing of x and x'' where x'' is, say, a random input. In the latter, there will be little correlation. So Definition 3 is not satisfied.

3.2 Format Preserving Encryption - the Cycle Walking Technique

A good example of a construction that is inherently non fixed-time, yet key oblivious and under some conditions query-oblivious, is the cycle walking technique of Format Preserving Encryption (See Bellare et al. [6]). Imagine that we have a construction of a pseudo-random permutation (PRP) on some domain, say of size 2^ℓ, and we want to build from it a PRP on a smaller domain $S \subset [2^\ell]$. A simple example is when S is the set of numbers 0 through $Q-1$ (where Q is not a power of 2), but the question is relevant for any format of S.

What Black and Rogaway [9] analyzed is a simple cycle walking technique for constructing permutations on smaller domains. Given a PRP π on the larger domain of size 2^ℓ, to get a PRP on S define π' by starting with $x \in S$ and repeatedly apply π to it until hitting a value in S. This is defined to be $\pi'(x)$. The expected number of applications of π is $2^\ell/|S|$. It is clear that this construction is *not* fixed-time (if $|S| < 2^\ell$), even if the original PRP is fixed-time. So which of the definitions of our framework does this technique satisfy? We note that Bellare et al. wrote that it "Doesn't Give Rise to Timing Attacks,"[3] but we want to check in what sense this is true and how it fits our definitions.

We claim that the exact properties of π' depend on whether the original PRP is fixed-time or "merely" key oblivious. For the former we get both key and query-obliviousness and for the latter just key obliviousness.

[2] In our case, for the ith distribution, the first $i-1$ timings are the application of G on one key (the real one, for which the output values are also given) and the last $m-i+1$ ones are the timings of another key, unrelated to the real one. If it is possible to distinguish between the first and last distributions, then it is possible to distinguish between some two neighboring distributions.

[3] What they proved is that leaking the number of applications of π does not hurt the pseudorandomness of π'

We assume that the implementation is such that the timing of the various calls to π are independent of each other, in the sense the time it takes to run $\pi(x)$ does not depend on any sequence of operations done before and in particular on whether we have just executed $\pi(x_1), \pi(x_2), \ldots, \pi(x_m)$.

Claim 5 *(i) If the permutation π is fixed-time, then the result π' is both key oblivious and query-oblivious in the weakly with results sense. (ii) If the permutation π is key oblivious, then the result π' is key oblivious.*

In order to see that (ii) is correct, think of simulating π' through access to π. Now if there is an attack on π' that distinguishes between true timing and timing of some random unrelated key, then we can apply it to π itself and get the same distinguishing probability, thus violating the assumption that π is key oblivious.

In order to see that (i) is true, think of composing the permutation π with a random permutation σ. By the definition of a PRP, this is indistinguishable from the plain π (since this is the case for a truly random permutation). Now instead of a query x, the query is effectively $\sigma(x)$, which makes it into a random unknown value. Given that the implementation is of π is fixed-time, the only information gained from the timing is the number of applications of π needed to evaluate $\pi'(x)$. But given the randomness of σ this is useless information to distinguish between two queries q'_0 and q'_1 that have not appeared so far and therefore the construction is query-oblivious.

Note that the construction does not satisfy the key switch requirement of Definition 2, since once π is fixed, then each element x has a very distinctive number of evaluations of π needed to compute $\pi'(x)$. Therefore it is possible to see if the time to compute $\pi'(x)$ changes at the potential key switch time.

3.3 Key-oblivious Domain Extension

Pseudo-random functions are a major cryptographic primitive that can be used to efficiently obtain many other primitives and is very useful in many protocols. One question that comes up is: given a function family F where each function $\mathcal{F}_k \in F$ is, say, length preserving, i.e. $\mathcal{F}_k \colon \{0,1\}^n \mapsto \{0,1\}^n$, how does one come up with constructions which are on larger domains, e.g. $\{0,1\}^{2n} \mapsto \{0,1\}^n$.

The Levin Trick: A common and simple way of obtaining domain extension is to apply a universal hash function from the larger domain to the smaller one and then apply the PRF. Namely, let Γ be family of pair-wise independent hash functions s.t. $g \colon \{0,1\}^{2n} \mapsto \{0,1\}^n$ for $g \in \Gamma$ and $\mathcal{F}_k$ be a PRF. Then the extended function is defined as $\mathcal{F}'_{k,g}(x_1, x_2) = \mathcal{F}_k(g(x_1, x_2))$.

Claim 6 *Even if $\mathcal{F}_k$ is key-oblivious and even if g is fixed time, the resulting construction may not be key-oblivious.*

Proof. Consider the case that $\mathcal{F}_k$ is very much *not* query oblivious. That is, the time to compute $\mathcal{F}_k(x)$ is x. Now suppose that h is defined by two values $a_1, a_2 \in GF[2^n]$ (chosen uniformly at random) and the function is $g(x_1, x_2) = a_1 x_1 + a_2 x_2$ where the computation is over $GF[2^n]$. Given a few examples of pairs (x_1^i, x_2^i) and the corresponding values $\mathcal{F}'_{k,g}(x_1^i, x_2^i)$ it is possible to reconstruct a_1 and a_2. Once this is done it is possible to find collisions with g, i.e. two pairs (x_1, x_2) and (x'_1, x'_2) s.t. $g(x_1, x_2) = g(x'_1, x'_2)$. For this pair we will have that $\mathcal{F}'_{k,g}(x_1, x_2) = \mathcal{F}'_{k,g}(x'_1, x'_2)$. Now if the timing of a random key is given, then this will not cause a collision in $\mathcal{F}'_{k,g}$ and therefore there is a way to distinguish correct and incorrect timings.

Note that this claim shows that key-obliviousness is not necessarily preserved under composition. In this example both $\mathcal{F}_k$ and g are key-oblivious, yet the composition is not.

Corollary 1. *There are keyed functions f and g such that given key-oblivious implementations of f and g the resulting composition $f(g(x))$ is not key oblvious.*

Using CRH and UOWHF: A way to remedy this problem is to use a Collision Resistant Hash (CRH) function, instead of a combinatorial one used in the original proposal by Levin. Recall that a function h is a CRH if it is hard to find two different values x and x' that collide under h, that is, any collision is considered a violation of the hardness assumption. But this approach (i) Requires another cryptographic assumption or primitive. Recall that in terms of assumptions, PRFs can be built from one-way functions in a black-box (BB) manner, whereas CRHs are BB separated from one-way functions. (ii) May not work with all range of the parameters we are interested in, since a CRH requires a minimum range size. For instance, the range of the CRH cannot be 80 bits.

The perhaps surprising observation is that we show that the CRH in the above construction can be replaced with a Universal One-way Hash Function (UOWHF) [32]. UOWHF (or second pre-image resistant hash function), are ones where the target x is chosen before the function is known and the collision should be with the target x. UOWHF can be based on one-way functions [34,22]. The idea is to replace each x_i with its PRF value. Let $h \in H$ where H is a UOWHF family. Consider the construction

$$\mathcal{F}'_{k,k',h}(x_1, x_2) = \mathcal{F}_k(h(\mathcal{F}_{k'}(x_1), \mathcal{F}_{k'}(x_2))).$$

Claim 7 *If the implementation of $\mathcal{F}$ is key-oblivious, then for* any *implementation of the hash function h chosen from a family of UOWHFs, the implementation of $\mathcal{F}'$ as defined above is key-oblivious.*

The only case we need to worry is if the adversary finds collisions under h. The hardness properties of h do not guarantee hardness of finding *any arbitrary* collision, but rather with one target specified in advance. The idea is that an adversary may ask various queries, and since we do *not* assume that $\mathcal{F}_k$ is query oblivious and we make no timing assumption regarding h, then the values of $\mathcal{F}_{k'}(x_1)$ are known to the adversary (i.e. they may leak through the computation of h).

Furthermore, the adversary can mix and match an x_1^i and x_2^j from different queries. Nevertheless, the values $\mathcal{F}_{k'}(x_b^i)$ look random to the adversary. Suppose an adversary makes m queries to the function $\mathcal{F}_{k'}$. At the end of the attack it finds a collision with some pair (x_1^i, x_2^j) as one of the inputs with a collision in h (if the colliding pair was never queried this is even better). Then we can use this adversary as a second pre-image finder: We select a random value $(y_1, y_2) \in \{0,1\}^{2n}$ as the target and then guess i and j and when x_1^i is given, we plug in y_1 as the value of $\mathcal{F}_{k'}(x_1^i)$ and similarly for x_1^j we give y_2 as the value of $\mathcal{F}_{k'}(x_2^i)$. From the adversary's point of view this looks like a 'normal' instance. Therefore the probability of selecting i and j correctly is $\Omega(1/m^2)$ times the probability that the adversary finds a collision.

Cascading: We show that the *cascading domain extension*, as analyzed by Bellare, Canetti and Krawczyk [5] actually preserves key-obliviousness. The length-doubling construction is

$$F'_k(x_1, x_2) = F_{F_k(x_1)}(x_2).$$

The straightforward implementation of F' is to first compute $F_k(x_1)$ then take the resulting value as the key k_2 to $F_{k_2}(x_2)$ where the time of the second is independent of the time is took compute the first one.

Claim 8 *If the implementation of F is key-oblivious, then the "straightforward" implementation of F' is key-oblivious.*

Proof. (Sketch) The main issue is that in this construction it is always clear what the query is. The keys, that keep changing, on the other hand, are not known. Therefore the Bellare et al. proof can be translated to this setting. Consider the experiment where instead of using the value $F_k(x_1^j)$ and random value is v_j is used for the next step (while making sure to use it consistently) but the timing produced is that of $\mathcal{T}(F_k(x_1^j))$ and $\mathcal{T}(F_{v_j}(x_2^j))$. If it is possible to distinguish between these two cases, then there is an attack on the key obliviousness of F. If not, then note that we can view the new construction as a concatenation of many functions, as argued at the beginning of this section.

Question 1. A major issue we did not resolve is how *not to lose the birthday bound*, as was done in Berman et al. [7] for domain extension without timing. Is it possible to get a similar result when timing is leaked?

4 Key-Oblivious PRPs on Small Domains

The question of how to generate Pseudo Random Permutations (PRP) has been extensively investigated for a few decades. Luby and Rackoff [25] (who defined the notion of PRP) showed how to get a PRP from a PRF. Their construction uses a Feistel network and there are many variants of it. In their work the security of the construction works provided the adversary makes at most $O(N^{\frac{1}{4}})$ queries, where N is the size of the domain. When the domain is not very big, such a security guarantee might not be enough, as $N^{\frac{1}{4}}$ can be a feasible amount of queries made. This is also true to refinements of the method, such as Naor and Reingold [30], who get to $N^{\frac{1}{2}}$. In such cases we might want the security to hold even if the adversary queries a constant fraction of the domain or even all of the domain but a constant number of elements.

When the domain is very small, it is possible to generate a fully random permutation, rather than a PRP. Yet this is undesirable in many cases, since the memory required to represent a random permutation is $\Omega(N \log N)$ bits. This memory requirement may be feasible for small enough N but is infeasible for medium-sized N (e.g. all credit card numbers or SSNs). We are left with the intermediate case of small, but not too small, domains, so that explicitly saving a permutation is infeasible, yet the security guarantees of Feistel network constructions might not suffice. As a concrete example think of the domain of credit card numbers (16 decimal digits). We refer to PRPs of this intermediate case as small-domain PRPs.

Small-domain PRPs are useful in a variety of application scenarios, e.g. cryptographic constructions, as Oblivious RAMs [17,36], for randomly reordering (permuting) a list of items. They can also be used to generate pseudorandom *unique* tokens (e.g., product serial numbers) in a specific format and to encrypt data in a small domain, such as encrypting a 9-digit social security number into another 9-digit number. Because of this, a small-domain PRP is also commonly referred to as a small-domain cipher or format-preserving encryption (FPE).[4] FPE has been a useful tool in encrypting financial and personal identification information, and transparently encrypting information in legacy databases.

In this section we focus on the small domain and show an interesting construction of PRPs on which is not fixed-time, yet is also secure under some of our definitions. A line of three works addressed this issue and showed efficient constructions for PRPs on small domains N with strong security guarantees, based on PRP or PRFs on large domains. The key observation in these works is one made by Moni Naor (see [35]), that if a card shuffling algorithm is *oblivious*, meaning that one can trace the trajectory of a card without attending to a lot of other cards in the deck, then it gives rise to a computationally feasible PRP. Therefore we can think of $[N]$ as a deck of cards of size N. All three works we describe start from this viewpoint on PRPs. The dominant computational resource in these works is the calls to a PRF on a large domain.

The first work, called "Swap-or-Not Shuffle" (SN) by Hoang, Morris and Rogaway [19] consists of a sequence of rounds that gradually shuffle the deck. In each round they consider a random matching of the cards in the deck that matches card X with card $X \oplus K$ (the randomness is over the choice of K). Additionally, for each matched pair $X, X \oplus K$ there is a random and independent bit b that decides whether to swap the matched pair of cards or not (these bits are also derived from the key). Hoang et al. [19] proved that applying the swap-or-not procedure $O(\log N)$ times and picking the matching index K for each round at random and independently suffices as long as at most $(1-\varepsilon)N$ queries were made for any fixed $\varepsilon > 0$ (notice that this ε affects multiplicatively the number of rounds of swap-or-not needed to achieve a PRP).

To implement this as a PRP the swap bits along the shuffle b, as well as the K of the matching should be produced by a PRF (which can be derived from a PRP on *large* domain such as AES). This gives a PRP that runs in a fixed time and remains secure as long as at most $(1-\varepsilon)N$ queries were made for any fixed $\varepsilon > 0$ (notice that this ε affects by a multiplicative the number of rounds of swap-or-not needed to achieve a PRP). This procedure is fixed-time provided that the PRFs and XOR operations are implemented in running time independent on the inputs and keys.

The second work, called "Mix and Cut" by Ristenpart and Yilek [33], aimed to improve on the number of queries that can be made while keeping the PRP secure. Ristenpart and Yilek introduced a construction of PRP which runs in a fixed-time of $O(\log^2 N)$ and achieves *full security*, meaning it remains secure even

[4] Note that the cycle walking technique mentioned in Section 3 does not solve the problem of constructing a small or medium size PRPs, since it needs a PRP of not much larger size to begin with.

if the entire domain is queried. The Mix and Cut shuffle works by mixing the deck, cutting it to two equal parts, and mixing each of them recursively using the Mix and Cut shuffle. In the paper, they also proved that if the shuffle before each cut mixes the cards well enough (which means the top half and bottom half is approximately a random partition of the cards), then this procedure achieves full security. They explicitly showed that the Swap-or-Not shuffle can be used with the ε from last paragraph being $\frac{1}{2}$ to give a fully secure PRP with the Mix and Cut construction.

The third work, called "Sometimes Recurse" (SR) shuffle by Morris and Rogaway [29], constructed a PRP which runs in *expected* time of $O(\log N)$ and achieves *full security*. Morris and Rogaway observed that when the Mix and Cut procedure is called, there is no need to mix both the top half and the bottom half of the deck. Since the top half looks almost uniform, it is enough to mix only the bottom half, therefore they suggested to only recurse on the bottom half of the deck, hence the name sometimes recurse. This construction allows an improvement on the $O(\log^2 N)$ of the Mix and Cut shuffle to an expected number of rounds which is $O(\log N)$.

The downside of SR is that it is no longer fixed-time, and in fact the running time is fully determined by the number of leading 1s in the output. Morris and Rogaway address this issue by stating that in a very common use of SR an adversary sees the outputs anyway, and so the running time doesn't give more information. However it is also the case that keyed functions, and in particular PRPs, are employed as a *subroutine of a larger system* (see for example [3]). In such cases the adversary no longer sees the output, and so the running time might leak valuable information that can harm the security of the system.

Consider for example if the PRP is used for storing or transmitting some piece of sensitive information like a vote or a credit card number in a way that should not reveal the correspondence between the customer or voter and the ciphertext. In this case the adversary can only get the runtime of the transaction (purchase, vote) which corresponds to the runtime of the PRP but does not have direct access to the results of the queries. Suppose now that the adversary does get to see after some time a *batch of such ciphertexts* and perhaps some other information about them (say their opening in case of votes, or whether the transaction was declined for credit cards). If the adversary knows how long each transaction took, then it can connect the ciphertext and the voter or customer and learn something it should not have learned. Moreover, the SR scheme is vulnerable to a *denial-of-service attack* where the attacker can easily assemble *without any query* many different ciphertexts that take a long time to decrypt (by picking those that have long prefixes of '1's).

Before analyzing the security of SR we need to specify what we mean by runtime of SR. We assume that SR construction uses the Swap-or-Not (SN) PRF for each shuffle and that the Swap-or-Not is implemented in a fixed-time manner. I.e. for any $x \in \{0,1\}^{\log N}$ we have that the running time of $SN_k(x)$ is independent of x and k but *is dependent* on the input length $\log N$. We denote this fixed running time of the the Swap-or-Not on $\log N$ bit inputs by $\mathcal{T}_{\log N}(SN) := \mathcal{T}(SN_k(x))$ for any x, k. Consequently, by linearity of composition, the running time of SR on the input x is $\mathcal{T}(SR_k(x)) = \sum_{i=0}^{j} \mathcal{T}_{(\log N)-i}(SN)$ where j is the number of leading 1's in the output, i.e. we assume commands are executed one after the other on a single unit without branch predictions. We show that the SR construction is not secure with respect to our definitions in Section 2.1.

Claim 9 *The SR construction is* not *key-oblivious, i.e. does not satisfy Definition 1 and hence also is* not *key-switch oblivious, i.e. does not satisfy Definition 2.*

Proof. Recall that by definition of SR, an adversary can determine the number of leading 1s in the output, from the running time and vice versa. This property of SR gives the following simple strategy:

1. Choose a single element x and query it to get: $SR_{k_0}(x)$ and $\mathcal{T}(SR_{k_b}(x))$.
2. Check the number of leading 1s in $SR_{k_0}(x)$, which is $\mathcal{T}(SR_{k_0}(x))$ and compare it to the running time $\mathcal{T}(SR_{k_b}(x))$ (number of calls to SN by SR on x).
3. If $\mathcal{T}(SR_{k_0}(x))$ is equal to $\mathcal{T}(SR_{k_b}(x))$, return 0, else return 1.

Observe that the distribution of the number of leading 1s is a $Geo(\frac{1}{2})$ distribution truncated at n. Also since the SR shuffle is a PRP, then with constant probability $\mathcal{T}(SR_{k_0}(x)) \neq \mathcal{T}(SR_{k_1}(x))$. This gives the adversary a constant advantage in the key-oblivious game (Definition 1), and therefore the SR PRP is not key-oblivious. Note that by choosing a polynomial number of inputs $x_1, \ldots, x_\ell$, the adversary can get a winning advantage that is exponentially close to 1.

Corollary 2. *By Theorem 2 SR is not* $(G_{\mathsf{PRP}}, \frac{1}{2})$*-time-secure.*

Claim 10 *The SR construction is* not *secure with respect to query-obliviousness (Definition 3) even in the weakly without results sense.*

Proof. Observe that in the definition there is no restriction on making queries in one direction, we assume both forward and inverse queries. By the construction of SR, in the inverse direction the running time is determined by the number of leading $1's$ *in the input*. This gives an adversary a very simple attack. First, make an inverse query on the string x which we define to be half $1's$ in the beginning and then half $0's$. Now for the challenge pick x_0 to be the all $0's$ string, and x_1 the all $1's$ string and make inverse queries. If the running time is faster than that on x, return $b = 0$; else return $b = 1$. This gives the adversary a winning probability of 1 in the game defined in Definition 3.

4.1 JSR: Constructing key-oblivious and query-oblivious PRPs

So far we have seen fixed-time constructions and a non fixed-time construction which is not secure under our definitions. It begs the question: are there (interesting) constructions that are not fixed-time, yet are also secure under some of our definitions? We now show a construction of "SR with a twist", which: (1) achieves the same expected run time of $O(\log N)$ up to a multiplicative factor of 2, (2) is not fixed-time, and (3) is secure with respect to Definition 1 and with respect to Definition 3 (assuming we have a fixed-time implementation of a PRF).

The Proposed Construction: Janus Sometimes Recurse The construction takes two independent (i.e. with two independent keys) copies of SR on the same domain $[N]$, where the permutations are denoted by π and σ and the keys by k^π and k^σ, and composes π with σ^{-1}, see Figure 2. This is similar to the approach that Maurer and Pietrzak [26] used to move from non-adaptive to adaptive PRPs. We call this construction "Janus Sometimes Recurse" (JSR):

Algorithm 1 $JSR(x)$ with keys k^π and k^σ

1: **return** $\sigma^{-1}(\pi(x))$

The term "Janus" comes from the Roman god who was depicted as having two faces, since both the directions (encryption and decryption) are forward looking.

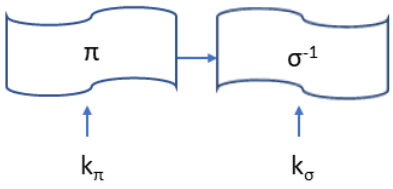

Fig. 2. Janus construction on two PRPs

The intuition for this construction is that while the running time of the forward direction leaks information about the output, the running time of the inverse is *determined by the input*, and so by composing the two we get that the running time of the algorithm both in the forward direction and in the inverse, is determined by *the inner value* which is *almost* independent of the input and output since π, σ are PRPs.

We now turn to show that (1) JSR is secure with respect to Definition 1 as well as Definition 3 when the two elements chosen are fresh (were not queried before), i.e., in the weakly sense, and (2) that it is not secure under Definition 2 even in the relaxed version when queries cannot repeat after the switch.

We start with a lemma that will help us prove the security of the construction.

Lemma 1. *Let π and σ be two PRPs on the same domain D, secure under q queries, and let (k_0^π, k_1^π) and (k_0^σ, k_1^σ) be two pairs of keys for π and σ respectively, then any PPT adversary* Adv *has a negligible advantage over $1/2$ in the following game:*

1. *A random bit $b \in_R \{0,1\}$ is chosen.*
2. *The adversary can make at most $j \leq q$ queries $x_1, \ldots, x_j$ to the composition of π with σ^{-1}, either in the forward direction $\sigma_{k_0^\sigma}^{-1} \circ \pi_{k_0^\pi}(x_i)$ or in the inverse direction: $\pi_{k_0^\pi}^{-1} \circ \sigma_{k_0^\sigma}(x_i)$.*
3. *The adversary gets in addition to the result of each query, the inner result with respect to b. Namely in the forward direction* Adv *gets $\pi_{k_b^\pi}(x_i)$ and in the inverse direction $\sigma_{k_b^\sigma}(x_i)$.*
4. *The adversary should output b' such that $b' = b$.*

Proof. Consider the distribution where a truly random value that hasn't appeared in previous queries is given instead of the inner value either in the case $b = 0$ or $b = 1$. We claim that the truly random distribution is indistinguishable from the distribution of inner values in both cases of b. Observe that the distribution of a truly random value that hasn't appeared in previous queries is the same distribution of the inner value of a composition of two truly random permutations that agree on the values already seen. Since π and σ are both PRPs, it follows that this distribution is indistinguishable from the inner value distribution both in $b = 0$ and $b = 1$. By a hybrid argument we get that the inner value distribution with $b = 0$ is indistinguishable from the inner value distribution with $b = 1$ which completes the proof.

Claim 11 *Assuming we have a fixed-time implementation of PRFs and the implementation of the JSR uses it, then the JSR construction is key-oblivious, i.e. secure with respect to Definition 1.*

Proof. The proof follows from Lemma 1 by observing that twice the number of leading 1's in $\pi_k(x)$ is exactly the running time for computing $\sigma_k^{-1} \circ \pi_k(x)$. Therefore the adversary can determine the running time of $\mathcal{F}_k(x)$ from $\pi_k(x)$. This implies that the advantage of the adversary in the key-oblivious game is at most the advantage of an adversary in the game defined in Lemma 1. Since we know that the advantage of an adversary in Lemma 1 is negligible, we get that the construction is secure with respect to Definition 1.

Claim 12 *The JSR construction is* not *key-switch secure. That is, it does not satisfy Definition 2 even in the relaxed version.*

Proof. Since we are interested in PRPs on small domains, this means that the adversary can query a large fraction or even *entire domain* apart from a few elements. The main observation is that the $\sum_{x \in D} \mathcal{T}(SR_{k_0}(x))$ is a constant independent of k_0, where the sum is over the running time of the JSR algorithm.

The strategy for the adversary is to query the entire domain up to a single element l (last element). By the observation above, this gives the adversary full information about the timing distribution of the last element. Now the switch happens and the adversary makes one last query on the remaining element. If the running time the adversary receives is the running time it expected with respect to k_0, then it returns $b = 0$; otherwise, it returns $b = 1$. Since the running time of $\mathcal{T}(SR_{k_1}(l))$ is not constant (with non-negligible probability), then with non-negligible probability $\mathcal{T}(SR_{k_0}(l)) \neq \mathcal{T}(SR_{k_1}(l))$ and so this strategy gives the adversary a non-negligible probability of winning.

Finally, we have:

Claim 13 *The JSR construction* is *query-oblivious secure, in the weakly and with results sense.*

Proof. The proof is similar to the proof of Claim 11. By a similar argument as the one in the proof of Lemma 1, we know that given the inner result of either q_0' or q_1', the adversary cannot tell which query it came from. Since given the inner result, the adversary can get the running time of the query, we conclude that given the running time $\mathcal{T}(SR_k(q_b'))$, the adversary cannot guess b.

We note that all claims above regarding the JSR construction hold to more general constructions. The crucial property of SR, from which the claims follow, is that *the running time of the algorithm was fully determined by the output*. We summarize:

Theorem 14. *Let π_k be a PRP for which the running time to compute π_k on x is fully determined by x's image, i.e. $\mathcal{T}(\pi_k(x)) = f(\pi_k(x))$ for some function f. If the permutation π is secure under q queries, then the Janus PRP $\pi_{k_0}^{-1} \circ \pi_{k_1}(x)$ is a key-oblivious and query-oblivious PRP secure under q queries.*

For an application of the above theorem to the low memory generation of a random permutation of the numbers in a given range $1 \ldots N$, see Appendix A.

5 Main Security Theorem, Noticeable Security and Games Definitions

5.1 Defining Noticeable Security and Cryptographic Games

Our goal is to prove that if the implementation of a keyed function is key-oblivious then the keyed function is time-secure, that is, the keyed function is secure w.r.t. the cryptographic game in which the adversary has access not only to the relevant oracles but also to the time it takes the oracle to execute. We show this for cryptographic keyed functions that have what we call **Noticeable Security.** Many well-known and useful keyed functions primitives have noticeable security, such as: (i) digital signatures, (ii) pseudo-random permutations (PRP), and (iii) indistinguishability of encryptions (either symmetric or not).

We start with defining a cryptographic game. We formulate a cryptographic game as an interactive game between a principal and an adversary. The principal maintains a state. In each round of an attack the adversary sends a pair (a_i, q_i) where q_i represents a query to the keyed function and a_i some additional information (for instance, a_i could be a bit indicating whether the adversary wishes to receive the upper half or the lower half of $\mathcal{F}_k(q_i)$). The principal responds with some function of its current state and the query. At the end, the adversary issues a guess and a tester decides whether to accept or not based on the state of the principal and the queries made (this determines who won the game).

Definition 6. *A **cryptographic game** $G_{\mathcal{F}}$ for a keyed function $\mathcal{F}_k(x)$ between a principal and an adversary is a game defined by three PPT algorithms*

$$(\mathsf{StateTransition}, \mathsf{Answer}, \mathsf{Tester}),$$

where the first two define the actions of the principal and Tester *produces outputs in $\{0, 1\}$ and determines who won the game. Specifically, at round i:*

- *Function* $\mathsf{StateTransition}$ *gets as input a state S_{i-1} and the current query pair issued by the adversary (a_i, q_i) and the value of $\mathcal{F}_k$ of q_i. The new state is S_i. In other words S_i is defined by:*

$$\mathsf{S}_i := \mathsf{StateTransition}(a_i, q_i, \mathsf{S}_{i-1}, \mathcal{F}_k(q_i)).$$

- Answer *takes as input the current state (S_i) and returns a response $\mathsf{CA_i}$ to the adversary (e.g. this may simply be $\mathcal{F}_k(q_i)$ or some function of it). That is*

$$\mathsf{CA_i} := \mathsf{Answer}(\mathsf{S}_i).$$

- Tester *gets as input all the queries pairs $(q_i, a_i)_{i=1}^{\ell}$ and states $(\mathsf{S}_i)_{i=1}^{\ell}$ as well the adversary's response* guess *and outputs either 0 or 1. If the output is 1 the adversary wins the game, otherwise it loses the game.*

For an adversary Adv*, the game $G_{\mathcal{F}}$ is the following:*

1. *A key $k \sim \mathcal{K}$ is chosen randomly.*
2. *Learning Phase: for $\ell = poly(n)$ rounds where at round i:*
   (a) Adv *chooses a query q_i that depends on $q_1, \ldots, q_{i-1}$ as well as $\mathsf{CA}_1, \ldots, \mathsf{CA}_{i-1}$.*
   (b) *The principal generates $\mathsf{S}_i := \mathsf{StateTransition}(a_i, q_i, \mathsf{S}_{i-1}, \mathcal{F}_k(q_i))$.*
   (c) *The principal sends* Adv *an answer CA_i which is a function of S_i and i.*
3. *Guessing Phase: The adversary generates* guess.
4. *Testing Phase: If $\mathsf{Tester}((a_1, q_1), \ldots, (a_\ell, q_\ell), \mathsf{S}_0, \mathsf{S}_1, \ldots, \mathsf{S}_\ell, \mathsf{guess}) = 1$, then* Adv *wins the game and otherwise it loses the game.*

Note that Tester does not know the key k, nor does it have query access to $\mathcal{F}_k$, hence the term **noticeable security**. Let $\tau < 1$ be a "benign" success probability. If the adversary cannot win the cryptographic game of a keyed function with probability much better than τ, then we say that the keyed function is noticeable secure. Specifically:

Definition 7. *Let $\tau < 1$, the "benign" success probability. A keyed function $\mathcal{F}_k(x)$ is $(G_{\mathcal{F}}, \tau)$-**noticeable-secure** if any PPT adversary* Adv *has probability at most $\tau + \mathsf{negl}(n)$ to win the game $G_{\mathcal{F}}$.*

As we shall see in Section 5.2, we can express in those terms many of the classical notions of security.

Adding Timing to the Game: Next, we define $(G_{\mathcal{F}}, \tau)$-time-secure which intuitively says that the winning probability in the game $G_{\mathcal{F}}$ remains $\tau + \mathsf{negl}(n)$ even when the adversary gets, not only the answers for the oracles queries, but also the time it takes for the oracle to execute them. More formally:

Definition 8. *Let $\tau < 1$, the "benign" success probability. The implementation of a keyed function $\mathcal{F}_k(x)$ is $(G_{\mathcal{F}}, \tau)$-**time-secure** if any PPT adversary* Adv *has probability at most $\tau + \mathsf{negl}(n)$ to win the game $G_{\mathcal{F}}$ with the following modification: in step 2.c of the game $G_{\mathcal{F}}$ as in Definition 6, the principal sends both CA_i and the time it takes for the principal to execute $\mathcal{F}_k$, that is $\mathcal{T}(\mathcal{F}_k(q_i))$.*

5.2 Classical Security Definitions are Noticeable Secure

We show a few examples of classical security definitions of keyed functions and show how they can be stated in the form of Definition 6, namely we show they are noticeable secure. Specifically, we show for digital signatures, pseudo-random permutations, and indistinguishability of encryptions.

Definition 9. *(Digital signatures) Let* Adv *be an adversary in the following game:*

1. *The principal generates public and secret keys (pk, sk) and shares pk with adversary* Adv.
2. *Adversary* Adv *chooses adaptively $\ell = poly(n)$ messages $m_1, \ldots, m_\ell$ and gives them to the principal, receiving their signatures*
$$\sigma_1 = \mathsf{Sign}_{sk}(m_1), \ldots, \sigma_\ell = \mathsf{Sign}_{sk}(m_\ell).$$
3. *Adversary* Adv *chooses a new message $m' \notin \{m_1, \ldots, m_\ell\}$. Adversary* Adv succeeds *if and only if it can generate $\sigma' = \mathsf{Sign}_{sk}(m')$ s.t. $\mathsf{Vrfy}_{pk}(m', \sigma') = 1$.*

A signature scheme $\Pi = (\mathsf{Sign}, \mathsf{Vrfy})$ is existentially unforgeable under an adaptive chosen-message attack, or just secure, if for all probabilistic polynomial-time adversaries Adv*, there is a negligible function* negl *such that:*
$$\Pr[\mathsf{Adv}\ \textit{success}] \leq \mathsf{negl}(n).$$

It is relatively straightforward to phrase the security of digital signatures in the game framework of Section 5.1. Let G_{DS} be the following cryptographic game:

1. The principal generates public and secret keys (pk, sk) and shares pk with adversary Adv.
$$\mathsf{S}_0 = pk$$

2. Learning phase: For $i \in [1, \ell]$,
   (a) Adv chooses adaptively a message m_i and gives it to the principal.
   (b) The principal is essentially stateless, i.e. its current state is simply the signature $\mathsf{Sign}_{sk}(m_i)$ on m_i.
   (c) The answer function is simply to send the full state to the adversary, i.e.

$$\mathsf{CA}_i = \mathsf{S}_i = \mathsf{Sign}_{sk}(m_i).$$

3. Guessing phase: Adversary Adv chooses a new message $m' \notin \{m_1, \ldots, m_\ell\}$ and calculates

$$\mathsf{guess} = (m', \mathsf{Sign}_{sk}(m')).$$

4. Testing phase: Tester returns 1 iff m' is not in $\{m_1, m_2, \ldots, m_\ell\}$ and $\mathsf{Vrfy}_{pk}(\mathsf{guess}) = 1$.

Proposition 1. *A digital signature scheme is secure according to Definition 9 iff it is $(G_{\mathsf{DS}}, 0)$-noticeable-secure.*

Notice that when discussing key-obliviousness for digital signatures the adversary also knows the corresponding public-key to k_0 (but not the one corresponding to k_1).

Definition 10. *(Strong Pseudo-random Permutations) Let Adv be a probabilistic polynomial-time adversary in the following game:*

1. *A key $k \sim \mathcal{K}$ is sampled.*
2. *A random bit b is sampled.*
3. *Adv chooses adaptively $\ell = poly(n)$ queries $(x_1, s_1), \ldots, (x_\ell, s_\ell)$ where x_i is a message and $s_i \in \{-1, 1\}$ is the oracle direction. If $b = 0$ then it receives the values of E_k on each query*

$$y_1 = E_k^{s_1}(x_1), \ldots, y_\ell = E_k^{s_\ell}(x_\ell).$$

   If $b = 1$ then it receives the values of a random permutation f on each query

$$y_1 = f^{s_1}(x_1), \ldots, y_\ell = f^{s_\ell}(x_\ell).$$

4. *Adversary Adv guesses b' and wins if $b' = b$.*

The function E_k is a pseudo-random permutation if for all PPT adversaries Adv as above there is a negligible function negl such that, for all n,

$$\Pr[b = b'] \leq 1/2 + \mathsf{negl}(n).$$

Note that a random permutation f can be simulated perfectly by always picking a random value that has not appeared so far (while being consistent with the values that have appeared).

As before, it is relatively straightforward to phrase the security of a PRP in terms of a game of Definition 6. Let the G_{PRP} be:

1. A random key $k \sim \mathcal{K}$ is chosen and the principal chooses a random bit b

$$\mathsf{S}_0 = b.$$

   The bit b remains in the state of the principal throughout the game.
2. Learning Phase: For $i \in [1, \ell]$:
   (a) Adversary Adv chooses adaptively a query (x_i, s_i) where x_i is a message and $s_i \in \{-1, 1\}$ is the direction of the permutation (forward or background).
   (b) The principal state includes the bit b and all the queries $(x_1, s_1, y_1), \ldots, (x_{i-1}, s_{i-1}, y_{i-1})$ values it sent and received. The state transition gets in addition to the current state the value of $E_k^{s_i}(x_i)$. If $b = 0$ then $y_i = E_k^{s_i}(x_i)$ and if $b = 1$ then, if x_i is equal to any of previous values x_j or y_j (with the appropriate s_j), then set y_i as was the previous response. Else choose y_i to be a random value that has not appeared so far.

(c) The answer

$$\mathsf{CA}_i = y_i.$$

3. Guessing Phase: Adversary Adv guesses $\mathsf{guess} := b'$.
4. Testing phase: returns 1 and wins if $\mathsf{guess} = b$.

Proposition 2. *A strong pseudo-random permutation is secure iff it is* $(G_{\mathsf{PRP}}, \frac{1}{2})$*-noticeable-secure.*

Our final example is indistinguishability of a private-key encryption scheme against chosen plaintext attacks (the same also works in public-key setting):

Definition 11. *(Indistinguishability of encryptions) Let* Adv *be a PPT adversary in the following game:*

1. *A random key* $k \sim \mathcal{K}$ *is chosen.*
2. *Adversary* Adv *chooses adaptively* $\ell = poly(n)$ *messages* $m_1, \ldots, m_\ell$ *and gives them to the principal, receiving their encryption in the form of ciphertext*

$$C_1 = E_k(m_1), C_2 = E_k(m_2), \ldots, C_\ell = E_k(m_\ell).$$

3. *Adversary* Adv *then chooses two more messages* $m'_0 \neq m'_1$ *such that* $|m'_0| = |m'_1|$ *(not necessarily distinct from the previous messages) and sends them to the principal.*
4. *The principal chooses randomly* $b \in_R \{0, 1\}$ *and sends* $E_k(m'_b)$ *back to adversary* Adv.
5. Adv *chooses adaptively another collection of* $\ell = poly(n)$ *messages* $m_{\ell+1}, \ldots, m_{\ell+\ell}$ *and receives their encryption in the form of ciphertexts*

$$C_{\ell+1} = E_k(m_{\ell+1}), \ldots, C_{\ell+\ell} = E_k(m_{\ell+\ell}).$$

6. *Adversary* Adv *guesses* b' *and wins if* $b' = b$.

A private-key encryption scheme is said to have the property of indistinguishablity of encryptions under a chosen-plaintext attack, if for all PPT adversaries Adv *as above there is a negligible function* negl *such that for all* n

$$\Pr[b = b'] \leq 1/2 + \mathsf{negl}(n).$$

It is relatively straightforward to phrase the security of indistinguishability of encryptions in the game framework of Section 5.1. Let G_{IND} be the following cryptographic game:

1. A random key $k \sim \mathcal{K}$ is chosen and the principal chooses a random bit b.

$$\mathsf{S}_0 = b.$$

2. Learning Phase:
   - For each $i \in [1, \ell - 1]$, adversary Adv chooses adaptively a message m_i and gives it to the principal, receiving its encryption in the form of ciphertext

$$\mathsf{CA}_i = \mathsf{S}_i = E_k(m_i).$$

   - Adversary Adv chooses two more messages $m_{\ell-1} \neq m_\ell$ such that $|m_{\ell-1}| = |m_\ell|$ and sends them to the principal. The principal calculates

$$\mathsf{S}_{\ell-1} = E_k(m_{\ell-1}) \text{ and } \mathsf{S}_\ell = (E_k(m_{\ell-1}), E_k(m_\ell), b).$$

$$\mathsf{CA}_{\ell-1} = \mathsf{NULL} \text{ and } \mathsf{CA}_\ell = (E_k(m_{\ell-1+b}),$$

   - For each $i \in [\ell + 1, 2\ell]$, adversary Adv chooses adaptively a message m_i and gives it to the principal, receiving its encryption in the form of ciphertext

$$\mathsf{CA}_i = \mathsf{S}_i = E_k(m_i).$$

3. Guessing phase: Adversary Adv guesses $\mathsf{guess} = b'$
4. Testing phase: returns 1 if $\mathsf{guess} = b$.

Proposition 3. *The indistinguishability of encryptions game* G_{IND} *is secure iff it is* $(G_{\mathsf{IND}}, 0.5)$*-noticeable-secure.*

5.3 Main Theorem: Key Oblivious Implies Time-Security

We now prove the main theorem of this section:

Theorem 15. *Let $\mathcal{F}_k$ be a keyed function that is $(G_{\mathcal{F}}, \tau)$-noticeable-secure for $\tau < 1$. If the implementation of $\mathcal{F}_k$ is key-oblivious, then the implementation of $\mathcal{F}_k$ is $(G_{\mathcal{F}}, \tau)$-time-secure.*

Proof. Recall that $G_{\mathcal{F}}$ is the cryptographic game of $\mathcal{F}_k$. We prove the theorem in two steps. We first consider a random-time game of $G_{\mathcal{F}}$, which we denote by $G^R_{\mathcal{F}}$. The random-time game $G^R_{\mathcal{F}}$ is similar to $G_{\mathcal{F}}$, with the following changes: (1) at the beginning of the game a random key k' is generated (unrelated to the original key k), and (2) besides receiving CA_i from the principal, the adversary receives the timing information $\mathcal{T}(\mathcal{F}_{k'}(q_i))$, which is the running time it takes for $\mathcal{F}_{k'}$ (namely, using the random key k') to be executed on q_i. It is easy to see that the winning probability of $G^R_{\mathcal{F}}$ is the same as the winning probability of $G_{\mathcal{F}}$. To see that, assume adversary $\mathsf{Adv}_{\mathsf{GR}}$ attacks $G^R_{\mathcal{F}}$. We build adversary $\mathsf{Adv}_{\mathsf{G}}$ attacking $G_{\mathcal{F}}$. Adversary $\mathsf{Adv}_{\mathsf{G}}$ works as follows: it first chooses a random key k', then for every query q_i from adversary $\mathsf{Adv}_{\mathsf{GR}}$, it calculates $\mathcal{T}(\mathcal{F}_{k'}(q_i))$ and sends CA_i together with $\mathcal{T}(\mathcal{F}_{k'}(q_i))$ to adversary $\mathsf{Adv}_{\mathsf{GR}}$. Finally, adversary $\mathsf{Adv}_{\mathsf{G}}$ returns the guess of adversary $\mathsf{Adv}_{\mathsf{GR}}$. From the description it holds that

$$\Pr[\mathsf{Adv}_{\mathsf{G}} \text{ wins } G_{\mathcal{F}}] = \Pr[\mathsf{Adv}_{\mathsf{GR}} \text{ wins } G^R_{\mathcal{F}}] \leq \tau + \mathsf{negl}.$$

Now, let $G^T_{\mathcal{F}}$ be the game defining the time-security of $\mathcal{F}_k$, namely it is similar to $G_{\mathcal{F}}$ but in addition to CA_i, the adversary also receives $\mathcal{T}(\mathcal{F}_k(q_i))$ from the principal. Let $\mathsf{Adv}_{\mathsf{GT}}$ be adversary attacking the game $G^T_{\mathcal{F}}$. We show that if the implementation of $\mathcal{F}_k$ is key-oblivious then

$$\Pr[\mathsf{Adv}_{\mathsf{GT}} \text{ wins } G^T_{\mathcal{F}}] - \Pr[\mathsf{Adv}_{\mathsf{GR}} \text{ wins } G^R_{\mathcal{F}}] \leq \mathsf{negl} \tag{5.1}$$

and this will imply

$$\Pr[\mathsf{Adv}_{\mathsf{GT}} \text{ wins } G^T_{\mathcal{F}}] \leq \tau + \mathsf{negl}.$$

To prove Equation 5.1, we assume there exists adversary $\mathsf{Adv}_{\mathsf{GT}}$ attacking the game $G^T_{\mathcal{F}}$ and we build adversary $\mathsf{Adv}_{\mathsf{KO}}$ attacking the key-obliviousness of $\mathcal{F}_k$. Adversary $\mathsf{Adv}_{\mathsf{KO}}$ works as follows:

- Adversary $\mathsf{Adv}_{\mathsf{KO}}$ receives polynomially many queries q_i from adversary $\mathsf{Adv}_{\mathsf{GT}}$ and sends them to the principal.
- Adversary $\mathsf{Adv}_{\mathsf{KO}}$ receives from the principal $\mathcal{F}_{k_0}(q_i)$ and $\mathcal{T}(\mathcal{F}_{k_b}(q_i))$ for a random bit b. Adversary $\mathsf{Adv}_{\mathsf{KO}}$ sends these $\mathcal{F}_{k_0}(q_i)$ and $\mathcal{T}(\mathcal{F}_{k_b}(q_i))$ to adversary $\mathsf{Adv}_{\mathsf{GT}}$.
- Adversary $\mathsf{Adv}_{\mathsf{GT}}$ returns its guess. Adversary $\mathsf{Adv}_{\mathsf{KO}}$ tests if the guess is correct using Tester and receives $w = \mathsf{Tester}(\mathsf{guess})$. If $w = 1$, then $\mathsf{Adv}_{\mathsf{KO}}$ returns $b' = 0$ since the queries were executed on $\mathcal{F}_{k_0}$. If $w = 0$, it returns $b' = 1$.

$$\begin{aligned}
\Pr[\mathsf{Adv}_{\mathsf{KO}} \text{ wins key-oblivious game}] &= \Pr[b' = 0 | b = 0] \cdot \Pr[b = 0] + \Pr[b' = 1 | b = 1] \cdot \Pr[b = 1] \\
&= \Pr[b' = 0 | b = 0] \cdot \Pr[b = 0] + (1 - \Pr[b' = 0 | b = 1]) \cdot \Pr[b = 1] \\
&= \Pr[\mathsf{Adv}_{\mathsf{GT}} \text{ wins } G^T_{\mathcal{F}}] \cdot 0.5 + (1 - \Pr[\mathsf{Adv}_{\mathsf{GR}} \text{ wins } G^R_{\mathcal{F}}) \cdot 0.5
\end{aligned}$$

Since $\Pr[\mathsf{Adv}_{\mathsf{KO}} \text{ wins key-oblivious game}] \leq 0.5 + \mathsf{negl}$, we get Equation 5.1.

We can use the theorem above, in addition to Propositions 1, 2 and 3, in order to obtain the following corollary:

Corollary 3. *The cryptographic schemes: digital signature, pseudo-random permutation and encryption are $(G_{\mathcal{F}}, \tau)$-time-secure with the corresponding τ for each game, provided the implementation of each scheme is key-oblivious.*

5.4 Caveat on Key-Oblivious and an Application of Query-Oblivious

It is essential not to abuse Theorem 15 and Corollary 3 and apply them correctly. The point is that the setting considered assumes that the timing information gained by the adversary follows the pattern of the timing game (Definition 6). But this may not necessarily be the case and it could be that the adversary does not know which query q is evaluated by the implementation of $\mathcal{F}_k$ at a given point and the time $\mathcal{T}(\mathcal{F}_k(q))$ it takes to compute $\mathcal{F}_k(q)$ may reveal information about q itself and compromise the security of the system.

A case in point is that of indistinguishably of encryption, where following a learning session an adversary selects a pair of messages (m_0, m_1) and receives either $\mathcal{F}_k(m_0)$ or $\mathcal{F}_k(m_1)$ and has to make a guess about the value of b where the ciphertext is of m_b. We saw in Proposition 3 that it falls into the framework of noticeable security. But this assumed that the principal computes both $\mathcal{F}_k(m_0)$ and $\mathcal{F}_k(m_1)$ before selecting which one to send. But a natural version of it is to first select whether to encrypt m_0 or m_1 and evaluate $\mathcal{F}$ only on the selected point. Thus the time to compute $\mathcal{F}_k(m_b)$ may reveal which one was selected. Note that here we have two games that are equivalent in the non-timed version (Definition 6), i.e. the adversary has exactly the same probability of winning in these two games, but when considering the timed version (Definition 8) these two games are not equivalent any more, since there function $\mathcal{F}_k$ is called after each query.

So given an implementation of a system, to argue that it is secure in terms of timing, one must argue that the setting of Definition 6 is the relevant one to the given system. Requiring the implementation of the encryption to be query-with-result-obliviousness solves the above situation in the indistinguishability of encryption game. If the implementation is query-with-result-oblivious, then the two games are equivalent also in the timed version:

Theorem 16. *Suppose $\mathcal{F}_k$ is a keyed function which is (G_{IND}, τ)-noticeable-secure. If the implementation of $\mathcal{F}_k$ is query-with-results-oblivious, then $\mathcal{F}_k$ is also (G_{IND}, τ)-time-secure.*

Proof. This is straightforward since the query-with-results-oblivious game is the exact same game as indistinguishability of encryptions with timing.

To conclude, arguing that the properties of key and query obliviousness are relevant to a given system is a delicate matter. Key obliviousness is most relevant to systems where the timing information is exposed for a certain period of time, but then when the attack is made, there is no leakage. Think of the definition of semantic security of encryption. If the attacker chooses a distribution on which it is to be tested, the encryption of the challenge message should be done after the leakage is over (note that it does not fall into the game framework, since there at any point it is clear what call is made to the keyed function).

On the other hand query obliviousness is relevant when the timing information is leaked about the message or challenge that is protected.

6 Future Work and Open Problems

In this paper we revisited issues arising from timing attacks and we investigated protecting keyed functions in computationally based cryptography, from timing attacks. To this end, we suggested several definitions, the most useful in the sense that implies much yet can be achieved is *key-oblivious*. We constructed small domain PRPs that are resilient to key-oblivious attacks, the *Janus Sometimes Recurse*. One natural question that comes up is:

Question 2. Can we lower the requirements from the underlying PRFs in the JSR construction? Can you work with PRFs that are not fixed-time?

The general issue is to understand when it is possible to replace fixed-time with key-oblivious primitives:

Question 3. Suppose we have a construction using keyed functions and we prove that it is key-oblivious assuming the keyed functions are fixed-time. When can we replace the keyed functions with key-oblivious implementation and maintain the overall key-oblivious property of the construction?

Another question is

Question 4. Is it possible to construct permutations on very large domains, while using a fixed-time implementation of the primitive which works only on small blocks? Recall that [30] suggested such a construction, but it loses a factor proportional to a birthday bound on the small domain.

Finally, can we get query-oblivious constructions from key-oblivious ones?

Question 5. Suppose we have a pseudo-random function that is key-oblivious. Can we use it in order to obtain one that is query-oblivious?

Finally, what is *composability* power of primitives that are *both* key-oblivious and query-oblivious? specifically PRFs?

Question 6. Suppose that we have a pseudo-random function that is both key and query-oblivious. Can we argue that any 'reasonable' construction using such components will be both key and query-oblivious. In particular, is this true for Feistel-based constructions, such as FF3?

References

1. José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, and Michael Emmi. Verifying constant-time implementations. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 53–70. USENIX Association, 2016.
2. Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Survey: Leakage resilience and the bounded retrieval model. In Kaoru Kurosawa, editor, *Information Theoretic Security, 4th International Conference, ICITS 2009, Shizuoka, Japan, December 3-6, 2009. Revised Selected Papers*, volume 5973 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2009. doi:10.1007/978-3-642-14496-7_1.
3. Yuriy Arbitman, Moni Naor, and Gil Segev. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 787–796. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.80.
4. Gustavo Banegas, Daniel J. Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková. CTIDH: faster constant-time CSIDH. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):351–387, 2021. URL: https://doi.org/10.46586/tches.v2021.i4.351-387, doi:10.46586/TCHES.V2021.I4.351-387.
5. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Pseudorandom functions revisited: The cascade construction and its concrete security. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 514–523. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548510.
6. Mihir Bellare, Thomas Ristenpart, Phillip Rogaway, and Till Stegers. Format-preserving encryption. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*, volume 5867 of *Lecture Notes in Computer Science*, pages 295–312. Springer, 2009. doi:10.1007/978-3-642-05445-7_19.
7. Itay Berman, Iftach Haitner, Ilan Komargodski, and Moni Naor. Hardness-preserving reductions via cuckoo hashing. *J. Cryptol.*, 32(2):361–392, 2019. doi:10.1007/s00145-018-9293-0.
8. Arnab Kumar Biswas, Dipak Ghosal, and Shishir Nagaraja. A survey of timing channels and countermeasures. *ACM Comput. Surv.*, 50(1):6:1–6:39, 2017. doi:10.1145/3023872.
9. John Black and Phillip Rogaway. Ciphers with arbitrary finite domains. In Bart Preneel, editor, *Topics in Cryptology - CT-RSA 2002, The Cryptographer's Track at the RSA Conference, 2002, San Jose, CA, USA, February 18-22, 2002, Proceedings*, volume 2271 of *Lecture Notes in Computer Science*, pages 114–130. Springer, 2002. doi:10.1007/3-540-45760-7_9.
10. Jonathan Bootle, Claire Delaplace, Thomas Espitau, Pierre-Alain Fouque, and Mehdi Tibouchi. LWE without modular reduction and improved side-channel attacks against BLISS. In *Advances in Cryptology - ASIACRYPT 2018, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part I*, volume 11272 of *Lecture Notes in Computer Science*, pages 494–524. Springer, 2018. doi:10.1007/978-3-030-03326-2_17.

11. David Brumley and Dan Boneh. Remote timing attacks are practical. In *Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003*. USENIX Association, 2003.
12. David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982*, pages 199–203. Plenum Press, New York, 1982. doi:10.1007/978-1-4757-0602-4_18.
13. Scott A. Crosby, Dan S. Wallach, and Rudolf H. Riedi. Opportunities and limits of remote timing attacks. *ACM Trans. Inf. Syst. Secur.*, 12(3):17:1–17:29, 2009. doi:10.1145/1455526.1455530.
14. Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 40–56. Springer, 2013. doi:10.1007/978-3-642-40041-4_3.
15. Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1857–1874. ACM, 2017. doi:10.1145/3133956.3134028.
16. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.
17. Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.
18. Shafi Goldwasser and Guy N. Rothblum. How to compute in the presence of leakage. *SIAM J. Comput.*, 44(5):1480–1549, 2015. doi:10.1137/130931461.
19. Viet Tung Hoang, Ben Morris, and Phillip Rogaway. An enciphering scheme based on a card shuffle. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2012. doi:10.1007/978-3-642-32009-5_1.
20. Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003 Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003. doi:10.1007/978-3-540-45146-4_27.
21. Yael Tauman Kalai and Leonid Reyzin. A survey of leakage-resilient cryptography. In Oded Goldreich, editor, *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 727–794. ACM, 2019. doi:10.1145/3335741.3335768.
22. Jonathan Katz and Chiu-Yuen Koo. On constructing universal one-way hash functions from arbitrary one-way functions. *IACR Cryptol. ePrint Arch.*, page 328, 2005. URL: http://eprint.iacr.org/2005/328.
23. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology - CRYPTO '96, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996. doi:10.1007/3-540-68697-5_9.
24. Richard J. Lipton and Jeffrey F. Naughton. Clocked adversaries for hashing. *Algorithmica*, 9(3):239–252, 1993. doi:10.1007/BF01190898.
25. Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2):373–386, 1988. doi:10.1137/0217022.
26. Ueli M. Maurer and Krzysztof Pietrzak. Composition of random systems: When two weak make one strong. In Moni Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 410–427. Springer, 2004. doi:10.1007/978-3-540-24638-1_23.
27. Boaz Menuhin and Moni Naor. Keep that card in mind: Card guessing with limited memory. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 107:1–107:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.107.
28. Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Moni Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296. Springer, 2004. doi:10.1007/978-3-540-24638-1_16.
29. Ben Morris and Phillip Rogaway. Sometimes-recurse shuffle - almost-random permutations in logarithmic expected time. In *Advances in Cryptology - EUROCRYPT 2014, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 311–326. Springer, 2014. doi:10.1007/978-3-642-55220-5_18.

30. Moni Naor and Omer Reingold. On the construction of pseudorandom permutations: Luby-rackoff revisited. *J. Cryptol.*, 12(1):29–66, 1999. doi:10.1007/PL00003817.
31. Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. *SIAM J. Comput.*, 41(4):772–814, 2012. doi:10.1137/100813464.
32. Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 33–43. ACM, 1989. doi:10.1145/73007.73011.
33. Thomas Ristenpart and Scott Yilek. The mix-and-cut shuffle: Small-domain encryption secure against N queries. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 392–409. Springer, 2013. doi:10.1007/978-3-642-40041-4_22.
34. John Rompel. One-way functions are necessary and sufficient for secure signatures. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 387–394. ACM, 1990. doi:10.1145/100216.100269.
35. Steven Rudich. Limits on the provable consequences of one-way functions. *PhD Thesis, University of California*, 1988.
36. Emil Stefanov, Elaine Shi, and Dawn Song. Towards practical oblivious ram. *arXiv preprint arXiv:1106.3652*, 2011.

A Low Memory Generation of Permutations Resistant to Timing Leakage

We now consider the motivating example given in the introduction. Recall that our goal is to generate a random permutation of the numbers in $1 \dots N$ (i.e. enumerate all the numbers one by one in a random order), while using very little memory.

While the way the question was posed in the introduction had no specific notion of randomness (of the permutation), one possibility is that the generated permutation should be indistinguishable from a random permutation, and a weaker version that it will be hard to guess the next value (again compared to a random permutation). Therefore, the requirement we make is that a guesser, trying to guess sequentially $\pi(1), \pi(2), ...$ would succeed in guessing the output of π at the same success rate as if π was chosen to be a truly random permutation (such a guesser should have $\approx \ln N$ correct guesses w.h.p.). Compare that to the scenario considered in [27], where a dealer generates some permutations of the value and a guesser has to guess them. In that work, the dealer was all-powerful and the guesser had limited memory, but now the situation is the other way around.

Definition 12. *Let $\mathcal{D}$ be a distribution on random permutations on $[N]$. And let* Guesser *be a randomized algorithm that gets an element of N^k (for some k) and returns an element of $[N]$. The success value of the guesser in round k is defined by*

$$\mathcal{V}_k(\mathsf{Guesser}) = \Pr_{f \sim \mathcal{D}}[\mathsf{Guesser}\,(f(1), \dots, f(k)) = f(k+1)] \tag{A.1}$$

The total success value of Guess *is:*

$$\mathcal{V}(\mathsf{Guesser}) = \sum \mathcal{V}_k(\mathsf{Guesser})$$

It is immediate that the value of any guesser against a truly random permutation is $\sum_{k=1}^{N} \frac{1}{k} \approx \ln N$. A possible answer to the original problem is to use a PRP π on a domain of size N that satisfies Definition 10. This can be done assuming one has more standard primitives such as a pseudo-random function. At step i the dealer outputs $\pi(i)$. As described in Section 4, the best construction of PRPs on medium-sized domains is the "Sometimes Recurse" due to Morris and Rogaway and uses $\Theta(\log N)$ calls on average to the pseudo-random function in expectation in order to evaluate $\pi(i)$. I.e. the amount of memory needed is that of a key to a pseudorandom function and the total time is $O(N \log N)$ calls to the underlying PRF. Any non-negligible increase in the guessing expectation can be used to distinguish the permutation from random.

But now, suppose that before each guess the guesser sees how long it took to generate the value $\pi(i)$. As discussed in Section 4.1, the time the Morris-Rogaway construction takes is proportional to the number of leading '0's in the output of $\pi(i)$. This suggests that the leaked running time may improve the guesser's performance.

Indeed, if the timing information is known to the guesser, then the guessing task is much easier and there is a guesser $\mathsf{Guesser}$ with $\mathcal{V}(\mathsf{Guesser}) = (\ln 2)/2 \ln^2 N - O(\ln N)$: The scheme essentially partitions the set of numbers $1, \ldots, N$ into $\log N$ bins depending on the prefix of 0's. Each one of them can be considered as a separate game (since we can find the possible output bin from the running time). In the jth the game there are 2^j numbers and the expected number of correct guesses is $\ln(2^j) = j \ln 2$. Altogether we get that

$$\mathcal{V}(\mathsf{Guesser}) \geq \sum_{j=1}^{\log N - 1} j \ln 2 = 1/2 \cdot \log^2 N \cdot \ln 2 - O(\ln N).$$

By applying the JSR construction (Algorithm 1) we get the best of both worlds. The total number of applications of the PRF when generating all N numbers will remain $O(N \log N)$ and the generated permutation will be indistinguishable from truly random permutation. In particular, any PPT algorithm $\mathsf{Guesser}$ will yield on expectation $\ln N$ correct guesses, even when the guesser gets the timing information. To see this, suppose that we apply it as a test for key-obliviousness. Since without the timing information the adversary cannot guess on expectation better than $\ln N$ in expectation, then given the timing information of an independent key (from the real one) does not change the expectation.

B Separating the Definitions

We start by comparing the various the three notions suggested in the definitions of Section2.1 and their general security. See Figure 3 for illustration. In order to show the separation we assume that fixed-time PRPs exist.

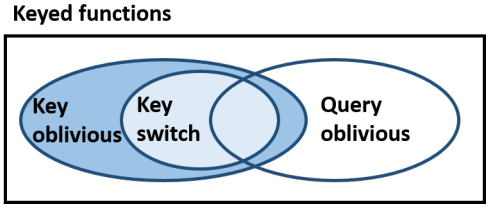

Fig. 3. Separating the definitions

We start by showing that Definition 1 and Definition 3 are incomparable definitions. This will also show that Definition 2 and Definition 3 are incomparable using the same constructions. To this end, we use pseudo-random permutations (PRP), formally defined in Definition 10 below.

Claim 17 *There exists a keyed function that is key-oblivious but not query-oblivious.*

Proof. Let $\mathcal{F}$ be a PRP with inputs of length n running in fixed time. Let $\mathcal{F}'$ be the same permutation as $\mathcal{F}$ only it runs for twice as long on the all 0s input. Clearly, $\mathcal{F}'$ is key-oblivious, since its running time on each input is the same no matter what the key is. However, querying the all 0s input, and any other input, it is easy to know which running time we got, which means it is not query-oblivious.

Claim 18 *There exists a keyed function that is query-oblivious but not key-oblivious.*

Proof. Let $\mathcal{F}$ be a PRP with inputs of length n running in fixed-time. Let $\mathcal{F}'$ be the same as $\mathcal{F}$ only it runs for twice as long on keys k where $\mathcal{F}_k(0)$ starts with 1 (roughly half the keys). Clearly, $\mathcal{F}'$ is query-oblivious, since its running time on each input is the same for the same key. However, to figure out with certain probability whether one is given the correct timing (that of k_0) or that of the other key k_1, query with 0 and see whether the result starts with 1 or 0 and whether the timing information is consistent with the result. With probability 1/2 the keys k_0 and k_1 have different outputs on input the all 0 string, and in these cases it is possible to distinguish the switch. Which means it is not key-oblivious.

Next, we show that Definition 2 is stronger than Definition 1 and we show the intersection of the three definitions:

Claim 19 *Any keyed function that is key-switch is also key-oblivious.*

Proof. If there is an adversary Adv, that has a non-negligible advantage in the game described in Definition 1, then there is an adversary Adv' that wins the game in Definition 2 by making no queries in step 2 in the key-switch game and simulating Adv' in step 4 of the game. Therefore Adv' has a non-negligible advantage in the game described in Definition 2.

Claim 20 *There exists a keyed function which is both key-oblivious and query-oblivious, but not key-switch.*

Proof. Let $\mathcal{F}$ be a PRP and consider the keyed function $\mathcal{F}'$ on the key space $\mathcal{K} \times \mathcal{K}$ such that $\mathcal{F}'_{(k_0,k_1)}(q) = \mathcal{F}_{k_0}(q)$ and $\mathcal{T}(\mathcal{F}'_{(k_0,k_1)}(q)) = c\mathcal{F}_{k_1}(q)$ for some c. Notice that $\mathcal{F}'$ is key-oblivious, as well as query-oblivious since $\mathcal{F}$ is a secure PRP.

Also, $\mathcal{F}'$ is not key-switch oblivious. Consider an adversary that queries half the domain in Definition 2 and then queries the other half of the domain with the key it got in the game after the switch. As the $\sum_q \mathcal{T}(\mathcal{F}'_{(k_0,k_1)}(q)) = \sum_q c\mathcal{F}_{k_1}(q)$ is some constant S independent of (k_0, k_1) (since $\mathcal{F}$ is a PRP). The adversary to the game of Definition 2 will compute the total computing time of $\mathcal{F}'$ before and after the switch, and will check if it is S; in this case, it will return that no switch was done.

Let $S_{\frac{1}{2}}$ be the distribution of the total running time of the first half of the domain (with measure induced by picking a random key from $\mathcal{K} \times \mathcal{K}$). Let X_1, X_2 be two independent samples from $S_{\frac{1}{2}}$, by construction the success probability of the algorithm is:

$$\mathcal{P}[X_1 + (S - X_2) = S] = \mathcal{P}[X_1 = X_2]$$

Since $\mathcal{F}$ is a PRP, the min-entropy of $S_{\frac{1}{2}}$ is vanishing as the key size grows to ∞, thus the success probability (i.e. the probability above) of the adversary in the game from Definition 2 goes to 1. See also the description of JSR in Section 4.1 for another example with similar analysis.

Claim 21 *There exists a keyed function that is both key-switch, key-oblivious, and query-oblivious.*

Proof. Let $\mathcal{F}$ be a PRP running in fixed-time for any input and for any key. The running time doesn't depend on the key therefore it is both key-oblivious and key-switch. The running time also doesn't depend on the input, therefore it is query-oblivious.

We remark that the assumption of fixed-time PRP is not needed for separating the definitions above. All proofs above hold under the assumption that $\mathcal{F}$ is fixed-time keyed function (not necessarily with cryptographic properties) on inputs of length n.