On the Power of Claw-Free Permutations

Yevgeniy Dodis* Leonid Reyzin†

August 2, 2002

Abstract

Probabilistic Signature Scheme (PSS), Full Domain Hash (FDH) and several of their variants are widely used signature schemes, which can be formally analyzed in the random oracle model. These schemes output a signature of the form $\langle f^{-1}(y), pub \rangle$, where y somehow depends on the message signed (and pub) and f is some public trapdoor permutation (typically RSA). Interestingly, all these signature schemes can be proven *asymptotically* secure for an *arbitrary* trapdoor permutation f, but their *exact* security seems to be significantly better for *special* trapdoor permutations like RSA. This leads to two natural questions: (1) can the asymptotic security analysis be improved with general trapdoor permutations?; and, if not, (2) what *general cryptographic assumption* on f — enjoyed by specific functions like RSA — is "responsible" for the improved security?

We answer both these questions. First, we show that if f is a "black-box" trapdoor permutation, then the poor exact security is unavoidable. More specifically, the "security loss" for general trapdoor permutations is $\Omega(q_{\mathsf{hash}})$, where q_{hash} is the number of random oracle queries made by the adversary (which could be quite large). On the other hand, we show that all the security benefits of the RSA-based variants come into effect once f comes from a family of *claw-free permutation* pairs. Our results significantly narrow the current "gap" between general trapdoor permutations and RSA to the "gap" between trapdoor permutations and claw-free permutations. Additionally, they can be viewed as the first *security/efficiency* separation between these basic cryptographic primitives. In other words, while it was already believed that certain cryptographic objects *can* be build from claw-free permutations but *not* from general trapdoor permutations, we show that certain important schemes (like FDH and PSS) provably work with *either*, but enjoy a much better tradeoff between security and efficiency when deployed with claw-free permutations.

*Department of Computer Science, New York University, 251 Mercer Street, New York, NY 10012, USA. Email: `dodis@cs.nyu.edu`

†Department of Computer Science, Boston University, 111 Cummington Street, Boston, MA 02215, USA. URL: `http://www.cs.bu.edu/~reyzin`

1 Introduction

FDH-LIKE SIGNATURE SCHEMES. In 1993, Bellare and Rogaway [BR93] formalized the well-known "hash-and-sign" paradigm for digital signature schemes by using the random oracle model. Specifically, they showed that if f is a trapdoor permutation and $\mathcal{RO}$ is a random function from $\{0,1\}^*$ to the domain of f, then signing a message m via $f^{-1}(\mathcal{RO}(m))$ is secure. This signature scheme was subsequently called "Full-Domain-Hash" or FDH.

In 1996, Bellare and Rogaway [BR96] pointed out that no tight security reduction from breaking FDH to inverting f was known. The best known security reduction lost a factor of $q_{\mathsf{hash}} + q_{\mathsf{sig}}$ in security (q_{hash} and q_{sig} represent the number of queries the forger makes to $\mathcal{RO}$ and to the signing oracle, respectively). This meant that the inverter could invert f with much lower probability than the probability of forgery. This in turn required one to make a stronger assumption on f, potentially increasing key size and losing efficiency.

To overcome this problem in case the trapdoor permutation is RSA (or Rabin), [BR96] proposed to hash the message with a random seed and format the result in a particular way (to fit it into the domain) before putting it through the permutation. The resulting scheme was thus probabilistic even when $\mathcal{RO}$ was fixed; it was termed PSS for "Probabilistic Signature Scheme." The security reduction for PSS (analyzed with RSA) is essentially lossless.

A natural question thus arose: was PSS necessary? In other words, could it be that a lossless security reduction for RSA-FDH was simply overlooked? In 2000, Coron [Cor00] found a better reduction for RSA-FDH that lost a factor of q_{sig} (instead of $(q_{\mathsf{hash}} + q_{\mathsf{sig}})$), suggesting that perhaps further improvements were possible. However, in 2002 Coron [Cor02] answered the question by showing that *any* black-box reduction for RSA-FDH had to lose a factor of at least q_{sig}, thus justifying the necessity of PSS. In the same paper, he also introduced a scheme called PFDH (for "Probabilistic Full-Domain Hash"), which signs m by computing $\langle \mathsf{RSA}^{-1}(\mathcal{RO}(m\|r)), r\rangle$ for a random r. This scheme is essentially PSS without the complicated formatting (and hence with slightly longer outputs), but with the same tight security.

FROM GENERIC ASSUMPTION TO RSA. While in 1993 [BR93] FDH was introduced to work with *any* trapdoor permutation, in 1996 [BR96] PSS, and in 2002 [Cor02] PFDH were considered only for RSA. Moreover, in 2002 [Cor00] the improved security reduction for FDH was only shown with RSA as well. This shift from generic assumptions to specific ones, while motivated by practical applications of the constructions, obscured what it was *exactly* about RSA that made PSS and PFDH reductions nearly tight, and accounted for the better security reduction for FDH. It is beneficial to consider which properties of RSA are crucial here, and which are merely incidental. Among other potential insights to be gained from such a consideration, is the possibility of using FDH, PSS or PFDH with other permutations. To emphasize this point and to avoid further confusion, we will denote by FDH, PFDH and PSS the signature schemes above considered with *general* trapdoor permutation f, and by RSA-FDH, RSA-PFDH, RSA-PSS the specific variants with f being RSA.

OUR CONTRIBUTION. We consider the question of identifying the *general* cryptographic assumptions that make the aforementioned efficient security reductions possible. Because all these schemes can be easily proven *asymptotically* secure with *any* trapdoor permutation f, it is natural to consider whether trapdoorness of f is the only security assumption necessary for a *tight* security reduction. The answer is "no": we show that a tight security reduction is *impossible* for FDH, PFDH, PSS, and in fact any scheme that consists of applying f^{-1} to the result of applying the random oracle to the message (possibly formatted with some randomness), if the scheme is to be analyzed with a general "black-box" trapdoor permutation f. Moreover, any black-box security reduction for such schemes has to lose a factor of q_{hash} with a generic black-box trapdoor permutation f: thus, the current security analysis for generic FDH, PFDH and PSS cannot be improved.

We also show that the *general* cryptographic assumption that makes the security proof for PSS/PFDH tight, and the improved security proof for FDH [Cor00] work, is the assumption of *claw-free permutations*. In other words, while all three schemes are asymptotically secure with any trapdoor permutation f, the exact security is dramatically improved once f comes from a family of claw-free permutations (and the bounds are exactly the same as one gets with RSA)! We remark that from a technical point, the proof of security with general claw-free permutations is going to be almost completely identical to the corresponding proof with RSA. Indeed, we are not claiming that we found a new *proof*. Instead, our goal is to find an elegant *general assumption on f* so that *essentially the same* (in fact, conceptually simpler!) proof works.[1]

Claw-Free vs. Trapdoor. Our results also shed new light on the relationship between claw-free and trapdoor permutations. So far, it was already believed that the existence of claw-free permutations is a strictly stronger complexity assumption than the existence of trapdoor permutations. For example, it is known how to construct collision-resistant hash functions (CRHF) [Dam87] and (non-interactive) trapdoor commitments [KR00] based on claw-free permutations, but no such constructions from trapdoor permutations seem likely. In fact, Simon [Sim98] showed a black-box separation between the existence of one-way permutations and the existence of CRHF's, and his result seems to extend to trapdoor permutations. Coupled with the above-mentioned construction of CRHF from claw-free permutations, we get a plausible separation between the *existence* of claw-free permutations and trapdoor permutations.

Our results provide another, quite different way to separate claw-free permutations from trapdoor ones. Namely, instead of showing that something constructible with claw-free permutations (e.g., CRHF) is *not* constructible with trapdoor ones, we show that claw-free permutations can be *provably more efficient* (or, equivalently, more secure) for some constructions (e.g., those of FDH-like signature schemes) than trapdoor ones. In other words, a stronger assumption provides better *exact security/efficiency* than a weaker one, even though *both* of them work asymptotically. To the best of our knowledge, this is the first separation of the above form, where a stronger primitive is provably shown to improve security of a scheme already working with a slightly weaker primitive.

A Word on Black-Box Lower Bounds. We briefly put our black-box separation in relation to existing black-box lower bounds. Originating with the work of Impagliazzo and Rudich [IR89], several works (e.g., [Sim98, GMR01]) showed *impossibility* of constructing one primitive from another. In contrast, several other works (e.g. [KST99, GT00, GGK02]) showed the *efficiency limitations* of constructing one cryptographic primitive from another. On the other hand, the recent work of Coron [Cor02] showed that the existing *security analysis* of a certain useful scheme cannot be improved, if the reduction accesses the adversary in a black-box way. In our work, we show that the existing security analysis cannot be improved when based on a *general* complexity assumption, even though it can be improved in (quite general, but still) *special* cases.

Organization. In Section 2 we give the necessary background about trapdoor and claw-free permutation families, and the "FDH-like" signature schemes we consider. In Section 3 we show that claw-free permutations give all the advantages of RSA in terms of exact security and efficiency. In Section 4 we explain the "black-box model" and show that poor exact security of FDH-like signatures is inevitable

[1] A good analogy could be to look at the famous Cauchy-Schwartz inequality in mathematics stating $||a|| \cdot ||b|| \geq \langle a, b\rangle$. This inequality was originally proved for the case of real vectors under euclidean norm. However, once one generalizes this proof to arbitrary Hilbert spaces, the proof actually becomes more transparent, since one does not get distracted by the specifics of real numbers, concentrating only on the essential properties of inner product spaces.

when the security reduction treats the trapdoor permutation f and the adversary in a black-box manner. In Section 5 we show several constructions of claw-free permutation families based on stronger kinds of trapdoor permutations, further closing the "gap" between general claw-free permutations and the specific ones like RSA.

2 Definitions

We let PPT stand for probabilistic polynomial time, and $\mathsf{negl}(k)$ refer to some negligible function in the security parameter k. The *random oracle* model assumes the existence of a publicly accessible truly random function $\mathcal{RO} : \{0,1\}^* \to \{0,1\}$. Using trivial encoding tricks, however, we can always assume that $\mathcal{RO}(\cdot)$ will return as many truly random bits as we need in a given application.

2.1 Trapdoor and Claw-free Permutations

Definition 1 *A collection of permutations $\mathcal{F} = \{f_i : D_i \to D_i \mid i \in I\}$ over some index set $I \subseteq \{0,1\}^*$ is said to be a family of* trapdoor permutations *if:*

- *There is an efficient sampling algorithm $\mathsf{TD\text{-}Gen}(1^k)$ which outputs a random index $i \in \{0,1\}^k \cap I$ and a* trapdoor information TK.
- *There is an efficient sampling algorithm which, on input i, outputs a random $x \in D_i$. We write $x \leftarrow D_i$ as a shorthand for running this algorithm.*
- *Each f_i is efficiently computable given index i and input $x \in D_i$.*
- *Each f_i is efficiently invertible given the trapdoor information TK and output $y \in D_i$. Namely, using TK one can efficiently compute (unique) $x = f_i^{-1}(y)$.*
- *For any probabilistic algorithm A, define the advantage of A as*

$$\mathsf{Adv}_A^{\mathcal{F}}(k) = \Pr[x' = x \mid (i, \mathsf{TK}) \leftarrow \mathsf{TD\text{-}Gen}(1^k),\ x \leftarrow D_i,\ y = f_i(x),\ x' \leftarrow A(i,y)] \quad (1)$$

  *A is said to $(t(k), \varepsilon(k))$-*break *$\mathcal{F}$, if A runs in time at most $t(k)$ and $\mathsf{Adv}_A^{\mathcal{F}}(k) \geq \varepsilon(k)$. $\mathcal{F}$ is said to be $(t(k), \varepsilon(k))$-*secure *if no adversary A can $(t(k), \varepsilon(k))$-break it. In the asymptotic setting, we require that the the advantage of any* PPT *A is negligible in k. Put differently, f_i is hard to invert without the trapdoor TK.*

A classical example of a trapdoor permutation family is RSA, where $\mathsf{TD\text{-}Gen}(1^k)$ picks two random $k/4$-bit primes p and q, sets $n = pq$, $\varphi(n) = (p-1)(q-1)$, picks random $e \in \mathbb{Z}^*_{\varphi(n)}$, sets $d = e^{-1} \bmod \varphi(n)$ and outputs $i = (n, e)$, $\mathsf{TK} = d$. Here $D_i = \mathbb{Z}^*_n$, $f_i(x) = x^e \bmod n$, $f_i^{-1}(y) = y^d \bmod n$. The RSA *assumption* states that this $\mathcal{F}$ is indeed a trapdoor permutation family.

Remark 1 *When things are clear from the context, we will abuse the notation (in order to "simplify" it) and write: f for f_i, D or D_f for D_i, $\mathcal{A}(f, \ldots)$ for $\mathcal{A}(i, \ldots)$, $f \leftarrow \mathcal{F}$ for $(i \leftarrow I \cap \{0,1\}^k;\ f = f_i)$, and $(f, f^{-1}) \leftarrow \mathsf{TD\text{-}Gen}(1^k)$ for $(i, \mathsf{TK}) \leftarrow \mathsf{TD\text{-}Gen}(1^k)$. Also, we will sometimes say that f by itself is a "trapdoor permutation".*

Definition 2 *A collection of pairs of functions $\mathcal{C} = \{(f_i : D_i \to D_i,\ g_i : E_i \to D_i) \mid i \in I\}$ over some index set $I \subseteq \{0,1\}^*$ is said to be a family of* claw-free permutations *if:*

- *There is an efficient sampling algorithm $\mathsf{CF\text{-}Gen}(1^k)$ which outputs a random index $i \in \{0,1\}^k \cap I$ and a* trapdoor information TK.
- *There are efficient sampling algorithms which, on input i, output a random $x \in D_i$ and a random $z \in E_i$. We write $x \leftarrow D_i$, $z \leftarrow E_i$ as a shorthand.*

- *Each f_i (resp. g_i) is efficiently computable given index i and input $x \in D_i$ (resp. $z \in E_i$).*
- *Each f_i is a permutation which is efficiently invertible given the trapdoor information* TK *and output $y \in D_i$. Namely, using* TK *one can efficiently compute (unique) $x = f_i^{-1}(y)$.*
- *Each g_i induces a uniform distribution over D_i when $z \leftarrow E_i$ and $y = g_i(z)$ is computed.*
- *For any probabilistic algorithm B, define the advantage of B as*

$$\mathsf{Adv}_B^{\mathcal{C}}(k) = \Pr[f_i(x) = g_i(z) \mid (i, \mathsf{TK}) \leftarrow \mathsf{CF\text{-}Gen}(1^k),\ (x,z) \leftarrow B(i)] \tag{2}$$

  *B is said to $(t(k), \varepsilon(k))$-*break *$\mathcal{C}$, if B runs in time at most $t(k)$ and $\mathsf{Adv}_B^{\mathcal{C}}(k) \geq \varepsilon(k)$. $\mathcal{C}$ is said to be $(t(k), \varepsilon(k))$-*secure *if no adversary B can $(t(k), \varepsilon(k))$-break it. In the asymptotic setting, we require that the the advantage of any* PPT *B is negligible in k. Put differently, it is hard to find a "claw" (x, z) (meaning $f_i(x) = g_i(z)$) without the trapdoor* TK.

We remark that the usual definition in fact assumes that $E_i = D_i$, each g_i is also a permutation, and the generation algorithm also outputs a trapdoor TK' for g_i. We do not need this extra functionality for our application, which is why we use our slightly more general definition.

We also remark that a claw-free permutation family $\mathcal{C} = \{(f_i, g_i)\}$ immediately implies that the corresponding family $\mathcal{F} \stackrel{\text{def}}{=} \{f_i\}$ is a trapdoor permutation family. Indeed, an inverter A for $\mathcal{F}$ immediately implies a claw-finder B for $\mathcal{C}$ who feeds A a random challenge $y = g_i(z)$, where $z \leftarrow E_i$: finding $x = f_i^{-1}(y)$ then implies that $f_i(x) = g_i(z)$. Moreover, the reduction is *tight*: $\mathsf{Adv}_B^{\mathcal{C}} = \mathsf{Adv}_A^{\mathcal{F}}$. We will say that the resulting trapdoor permutation family $\mathcal{F} = \mathcal{F}(\mathcal{C})$ is *induced* by $\mathcal{C}$.

Several examples of claw-free permutations will be given in Section 5. We briefly mention just one example based on RSA. $\mathsf{CF\text{-}Gen}(1^k)$ runs $\mathsf{TD\text{-}Gen}(1^k)$ to get (n, e, d), also picks a random $y \in \mathbb{Z}_n^*$, sets $i = (n, e, y)$, $\mathsf{TK} = d$ and $f_i(x) = x^e \bmod n$, $g_i(z) = yz^e \bmod n$. Finding a claw (x, z) implies $y = (x/z)^e \bmod n$, which implies inverting RSA on a random input y. Thus, this family is claw-free under the RSA assumption. Notice, the induced trapdoor permutation family is exactly the regular RSA family.

Remark 2 *When things are clear from the context, we will abuse the notation (in order to "simplify" it) and write: f/g for f_i/g_i, D/E or D_f/E_g for D_i/E_i, $\mathcal{A}(f, g, \ldots)$ for $\mathcal{A}(i, \ldots)$, $(f, g) \leftarrow \mathcal{C}$ for $(i \leftarrow I \cap \{0,1\}^k;\ f = f_i;\ g = g_i)$, and $(f, f^{-1}, g) \leftarrow \mathsf{CF\text{-}Gen}(1^k)$ for $(i, \mathsf{TK}) \leftarrow \mathsf{CF\text{-}Gen}(1^k)$. Also, we will sometimes say that f by itself is a "claw-free permutation", and (f, g) is a "claw-free pair".*

2.2 Full Domain Hash and Related Signature Schemes

We first define the notion of a signature scheme and its security, and then describe the specific schemes considered in this paper.

SYNTAX. A signature scheme consists of three efficient algorithms: $\mathcal{S} = (\mathsf{Sig\text{-}Gen}, \mathsf{Sig}, \mathsf{Ver})$. $\mathsf{Sig\text{-}Gen}(1^k)$, where k is the security parameter, outputs a pair of keys $(\mathsf{SK}, \mathsf{VK})$. SK is the signing key, which is kept secret, and VK is the verification key which is made public. The randomized signing algorithm Sig takes as input a key SK and a message m from the associated message space $\mathcal{M}$, internally flips some coins and outputs a signature σ; we write $\sigma \leftarrow \mathsf{Sig}_{\mathsf{SK}}(m)$. We will usually omit SK and write $\sigma \leftarrow \mathsf{Sig}(m)$. The deterministic verification algorithm Ver takes as input the message m, the signature σ, the public key VK, and outputs the answer a which is either $\mathsf{succeed}$ (signature is valid) or fail (signature is invalid). We require that $\mathsf{Ver}(m, \mathsf{Sig}(m)) = \mathsf{succeed}$, for any $m \in \mathcal{M}$.

In case a signature scheme (like most of the schemes considered in this paper) is build in the random oracle model, we allows both Sig and Ver to use the random oracle $\mathcal{RO}$.

Security of Signatures. The security of signatures addresses two issues: what we want to achieve (security goal) and what are the capabilities of the adversary (attack model). In this paper we will talk about the the most common security goal: *existential unforgeability* [GMR88], denoted by UF. This means that any PPT adversary C should have a negligible probability of generating a valid signature of a "new" message. To clarify the meaning of "new", we will consider the following two attack models. In the *no message attack* (NMA), C gets no help besides VK. In the *chosen message attack* (CMA), in addition to VK, the adversary C gets full access to the signing oracle Sig, i.e. C is allowed to query the signing oracle to obtain valid signatures $\sigma_1, \ldots, \sigma_n$ of arbitrary messages $m_1, \ldots, m_n$ adaptively chosen by $\mathcal{A}$ (notice, NMA corresponds to $n = 0$). C is considered successful only if it forges a valid signature σ of a message m not queried to signing oracle: $m \notin \{m_1 \ldots m_n\}$. We denote the resulting asymptotic security notions by UF-NMA and UF-CMA, respectively. Quantitively, we define

$$\mathsf{Adv}_C^{\mathcal{S}}(k) = \Pr[\mathsf{Ver}_{\mathsf{VK}}(m, \sigma) = \mathsf{succeed} \mid (\mathsf{SK}, \mathsf{VK}) \leftarrow \mathsf{Sig\text{-}Gen}(1^k),\ (m, \sigma) \leftarrow C^{\mathsf{Sig}_{\mathsf{SK}}(\cdot)}(\mathsf{VK})] \quad (3)$$

(where m should not be queried to the signing oracle $\mathsf{Sig}(\cdot)$). Of course, in the random oracle model the adversary is also given access to the random oracle $\mathcal{RO}$.

Definition 3 *The adversary C is said $(t(k), q_{\mathsf{hash}}(k), q_{\mathsf{sig}}(k), \varepsilon(k))$-*break *$\mathcal{S}$, if C runs is time at most $t(k)$, makes at most $q_{\mathsf{hash}}(k)$ hash queries to $\mathcal{RO}$, $q_{\mathsf{sig}}(k)$ signing queries to $\mathsf{Sig}(\cdot)$, and $\mathsf{Adv}_C^{\mathcal{S}}(k) \geq \varepsilon(k)$. $\mathcal{S}$ is said to be $(t(k), q_{\mathsf{hash}}(k), q_{\mathsf{sig}}(k), \varepsilon(k))$-*secure, *if no adversary C can $(t(k), q_{\mathsf{hash}}(k), q_{\mathsf{sig}}(k), \varepsilon(k))$-break it. In asymptotic terms, $\mathcal{S}$ is* UF-CMA*-secure if $\mathsf{Adv}_C^{\mathcal{S}}(k) = \mathsf{negl}(k)$ for any PPT C, and* UF-NMA*-secure if $\mathsf{Adv}_C^{\mathcal{S}}(k) = \mathsf{negl}(k)$ for any PPT C which does not make any signing queries.*

FDH-like Schemes. The random oracle based signatures we will consider all have the following simple form. The verification key will be the description of some trapdoor permutation f, the secret key is the corresponding inverse f^{-1}. To sign a message m, the signer first transforms m into a pair $(y, pub) \leftarrow T(m)$ (where pub could be empty). This transformation T only utilizes the random oracle (and possibly some fresh randomness), but not anything related to f or f^{-1}. It also has the property that one can easily verify the validity of the triple (m, y, pub). Then the signer computes $x = f^{-1}(y)$ and returns $\sigma = (x, pub)$. On the verifying side, one first computes $y = f(x)$ and then verifies the validity of the triple (m, y, pub). Of course, for the resulting signature to be secure, the transformation T should satisfy some additional properties. Intuitively, the value y should be random (since f is hard to invert on random inputs) for every m, and also "independent" for different m's (more or less, this way the inverse of $y(m)$ should not give any information about the inverse of $y(m')$). Transformations T satisfying the above informally stated properties do not seem to exist in the standard model, but are very easy to come up with in the random oracle model.

Below we describe three very popular signature schemes of the above form: Full Domain Hash (FDH [BR93]), Probabilistic Full Domain Hash (PFDH [Cor02]), and Probabilistic Signature Scheme (PSS [BR96]). We remark that another family of similar schemes is described in [MR02]. These signatures utilize the "swap" method, and are designed for the purpose of improving the exact security of several Fiat-Shamir based signature schemes [FS86, GQ88, OS90, Mic94]. However, one can observe that the resulting signature schemes can be all viewed as less efficient and more complicated variants of the PFDH scheme, so we do not describe them. In the following, f is a trapdoor permutation with domain D, public key is f and secret key is f^{-1}.

FDH: $\mathsf{Sig}(m)$ returns $\sigma = f^{-1}(\mathcal{RO}(m))$, and $\mathsf{Ver}(m, \sigma)$ checks if $f(\sigma) = \mathcal{RO}(m)$ (we assume that $\mathcal{RO}$ returns a random element of D, by implicitly running the corresponding sampling algorithm with the randomness returned by $\mathcal{RO}$).

PFDH: This signature is parameterized by the length parameter k_0. $\mathsf{Sig}(m)$ picks a random $r \in \{0,1\}^{k_0}$ and returns $\sigma = \langle f^{-1}(\mathcal{RO}(m\|r)), r\rangle$, and $\mathsf{Ver}(m,\sigma)$ checks if $f(\sigma) = \mathcal{RO}(m\|r)$ (again, we assume that $\mathcal{RO}$ returns a random element of D). Notice that FDH is a special case with $|r| = k_0 = 0$. In general, we will see that the length of "salt" r plays a crucial role in the exact security of PFDH.

PSS: This signature is parameterized by two length parameters k_0 and k_1. For convenience, we will assume that it takes between $n-1$ and n bits to encode an element of D, so that every $(n-1)$-bit number is a valid element of D (this is not crucial, but makes the description simpler). We also syntactically split the random oracle $\mathcal{RO}$ into three independent random oracles $H : \{0,1\}^* \to \{0,1\}^{k_1}$, $G_1 : \{0,1\}^{k_1} \to \{0,1\}^{k_0}$ and $G_2 : \{0,1\}^{k_1} \to \{0,1\}^{n-k_0-k_1-1}$. Then, $\mathsf{Sig}(m)$ picks a random salt $r \in \{0,1\}^{k_0}$, computes $w = H(m\|r)$, $r^* = G_1(w) \oplus r$ and returns $\sigma = f^{-1}(0\|w\|r^*\|G_2(w))$. The verification $\mathsf{Ver}(m,\sigma)$ computes $y = f(\sigma) \in D$, splits $y = b\|w\|r^*\|\gamma$, recovers $r = G_1(w) \oplus r^*$, and accepts if $H(m\|r) = w$, $G_2(w) = \gamma$ and $b = 0$.

We remark that all these schemes makes sense for arbitrary trapdoor permutation families. To emphasize a specific family $\mathcal{F}$, we sometimes write $\mathcal{F}$-FDH, $\mathcal{F}$-PFDH, $\mathcal{F}$-PSS. In particular, when $\mathcal{F}$ = RSA, we get 3 specific schemes RSA-FDH, RSA-PFDH, RSA-PSS.

Security of FDH-like Signatures. All the FDH-like signatures above can be shown asymptotically UF-CMA-secure for arbitrary trapdoor permutation family $\mathcal{F}$. Unfortunately, in terms of *exact* security, the situation is not entirely satisfactory. Specifically, if $\mathcal{F}$ is (t',ε')-secure, then one can show $(t, q_{\mathsf{hash}}, q_{\mathsf{sig}}, \varepsilon)$-security of any of the above signature schemes where roughly $t \sim t'$ and $\varepsilon \sim \varepsilon' q_{\mathsf{hash}}$. Put differently, in *all three schemes* above, the "generic" analysis loses a very large factor q_{hash} in terms of exact security. Intuitively, the security reduction has to "guess" which of the q_{hash} random oracle queries made by the hypothetical signature breaker is "relevant" for the final forgery, and respond to this query in a manner that will help inverting the trapdoor permutation f on a challenge y. In case this guess is unsuccessful, the forgery of the breaker is useless. Unfortunately, we will show in Section 4 that this large security loss is inevitable when dealing with arbitrary trapdoor permutations.

On the other hand, the situation is much better for the RSA-based examples of the above three schemes. Specifically, for RSA-FDH we one can get $t \sim t'$ and $\varepsilon = O(\varepsilon' q_{\mathsf{sig}})$ [Cor00] (and [Cor02] showed that this analysis cannot be improved for RSA). Namely, even though the reduction is still not tight (i.e., $\varepsilon \gg \varepsilon'$), the security loss is only a factor $q_{\mathsf{sig}} \ll q_{\mathsf{hash}}$. On the other hand, RSA-PFDH and RSA-PSS are *tight*, i.e. $t \sim t'$ and $\varepsilon \sim \varepsilon'$ (provided the salt length is somewhat larger than the very moderate quantity $\log q_{\mathsf{sig}}$ [Cor02]), which dramatically improves on the the generic security loss of q_{hash}.

To summarize, generic bounds for FDH, PFDH and PSS are much worse than the specific bounds when $\mathcal{F}$ = RSA. One of the main objectives of this paper was to try finding a general reason for this gap. Namely, to find a very general condition of $\mathcal{F}$, so that all the benefits of RSA come into effect with *any* $\mathcal{F}$ satisfying this condition. As we show next in Section 3, the needed condition is that $\mathcal{F}$ *is induced by some family $\mathcal{C}$ of claw-free permutations.* Indeed, we saw in Section 2.1 that RSA could be viewed as being induced by the some natural claw-free family, which explains much tighter exact security.

3 Claw-Free Permutations Yield Improved Security

Why claw-free permutations seem useful. The precise reason why claw-free permutations are very useful for FDH-like schemes will be obvious from the proof we present later. Here, however, we give some preliminary observations why claw-freeness seems to be relevant. First, assume we are not working in the random oracle model. The most basic signature scheme that comes to mind is $\sigma = f^{-1}(m)$, where f is a trapdoor permutation. Unfortunately, it is trivially forgeable since every σ is a valid signature of $m = f(\sigma)$. The next fix would be to utilize some function g and to output $\sigma = f^{-1}(g(m))$. Notice, finding a forgery (m, σ) for this signature is equivalent to finding m and σ such that $f(\sigma) = g(m)$, which exactly amounts to finding a claw (σ, m) for the function pair (f, g). Thus, the above signature scheme is UF-NMA-secure iff (f, g) comes from a pair of claw-free permutations! In fact, the first UF-CMA signature scheme [GMR88] was based on claw-free permutations (and a very non-trivial extension of the simple observation above), before more general UF-CMA constructions were obtained [BM88, NY89, Rom90].

Alternatively, let us return to the random oracle model and consider the FDH scheme (in fact, even more general PFDH scheme). The adversary C successfully forges a signature of some message if it can come up with σ and $\tau = m\|r$, such that $f(\sigma) = \mathcal{RO}(\tau)$. In other words, C has to find a claw (σ, τ) for the function pair $(f, \mathcal{RO})$! Of course, the family $\{(f, \mathcal{RO})\}$ is not a regular family of claw-free permutations, since the random oracle is not a regular function.[2] In the security proof, however, we may (and will in a second) simulate the random oracle by picking a random z and setting $\mathcal{RO}(\tau) = g(z)$. In this case, any forgery σ, τ by C will result in a claw (σ, z) for a "regular" claw-free pair (f, g).

Our Result. We show that all the security (and efficiency) benefits of RSA-FDH, RSA-PFDH and RSA-PSS over using general trapdoor permutation family $\mathcal{F}$ come into effect once $\mathcal{F}$ is induced by a family $\mathcal{C}$ of claw-free permutations. In particular, the security loss for FDH becomes only $O(q_{\mathsf{sig}})$, while the reductions for PFDH and PSS are essentially tight (once the salt length is at least $\log q_{\mathsf{sig}}$). For concreteness of the discussion, we concentrate on a representative case of PFDH. Very similar discussion holds for FDH and PSS. The theorem below contrasts our proof with claw-free permutations with a much looser (yet inevitably so) proof with general trapdoor permutations.

Theorem 1 (Security of PFDH)

(a) *Assume $\mathcal{C}$ is a claw-free permutation which is (t', ε')-secure, and let $\mathcal{F} = \mathcal{F}(\mathcal{C})$ be the induced trapdoor permutation family. Then $\mathcal{F}$-PFDH with salt length[3] $k_0 \geq \log q_{\mathsf{sig}}$ is $(t, q_{\mathsf{hash}}, q_{\mathsf{sig}}, \varepsilon)$-secure, where[4] $t = t' - (q_{\mathsf{hash}} + q_{\mathsf{sig}} + 1) \cdot \mathsf{poly}(k)$ and $\varepsilon = \varepsilon'/(1 - q_{\mathsf{sig}} 2^{-k_0})$, so that $\varepsilon \sim \varepsilon'$ (up to a small constant factor) when $k_0 > \log q_{\mathsf{sig}}$.*

(b) *In contrast, if $\mathcal{F}$ is a general (t', ε')-secure family of trapdoor permutations, then for any salt length k_0, $\mathcal{F}$-PFDH is only $(t, q_{\mathsf{hash}}, q_{\mathsf{sig}}, \varepsilon)$-secure, where $t = t' - (q_{\mathsf{hash}} + q_{\mathsf{sig}} + 1) \cdot \mathsf{poly}(k)$ and $\varepsilon = \varepsilon'(q_{\mathsf{hash}} + 1)$.*

Proof: We start with more interesting part (a). Let C be the forger for $\mathcal{F}$-PFDH which $(t, q_{\mathsf{hash}}, q_{\mathsf{sig}}, \varepsilon)$-breaks it. We construct a claw-finder B for $\mathcal{C}$. B gets the function pair (f, g) as an input. It makes f the public key for PFDH and gives it to C, keeping g for itself. It also prepares q_{sig} random elements $r_1 \dots r_{q_{\mathsf{sig}}} \in \{0, 1\}^{k_0}$ — these will be the salts of the messages it will sign for C. We call this initial list L (this list will shrink as we move along).

[2] We could extend the notion of claw-free permutations to the random oracle model, where we allow the function g (as well as the adversary) to depend on the random oracle. In this setting, for any trapdoor permutation f, the security of PFDH indeed implies that the pair $(f, \mathcal{RO})$ results in a family of such "oracle claw-free permutations" (again, with a large "security loss" q_{hash}). This is to be contrasted with the regular model, where the existence of trapdoor permutations is unlikely to imply the existence of claw-free permutations.

[3] As shown in [Cor02], the analysis can be extended even to $k_0 < \log q_{\mathsf{sig}}$, but the reduction stops being tight.

[4] Here $\mathsf{poly}(k)$ is a fixed polynomial depending on the time it takes to evaluate f and g in $\mathcal{C}$.

To respond to a hash query $m'\|r'$, we distinguish three cases. First, if the value is already defined, we return it. Else, if $r' \in L$, B picks and remembers a random x', and returns $\mathcal{RO}(m'\|r') = f(x')$ to C. Finally, if $r' \notin L$, B picks and remembers a random z', and returns $\mathcal{RO}(m'\|r') = g(z')$ to C.

If the forger makes a signature query m_i, we pick the next element r_i from the current list, and see if $\mathcal{RO}(m_i\|r_i)$ is defined. If so, it is equal to $f(x_i)$ for some x_i, so we return $\langle x_i, r_i\rangle$ as the signature of m_i. Else, we pick a random x_i, define $\mathcal{RO}(m_i\|x_i) = f(x_i)$ and return $\langle x_i, r_i\rangle$ to C. In either case, we remove r_i from the list L.

Eventually, (with probability ε) C will output a forgery $\langle x, r\rangle$ of some message m. Without loss of generality we assume that C asked the hash query $m\|r$ before (if not, B can do it for C; this increases the number of hash queries by one). If the answer was $g(z)$, we get that $f(x) = g(z)$, so B outputs the claw (x, z). Otherwise (the answer was $f(x)$), we did not learn anything, so B fails.

We see that the probability ε' that B finds a claw is $\varepsilon \Pr(E)$, where E is the event that the forgery corresponded to $g(z)$ rather than to $f(x)$, so that B does not fail. It remains to show that $\Pr(E) \geq 1 - q_{\mathsf{sig}} 2^{-k_0}$. We notice, however, that the only way that B will fail is if the value r was still in the list L at the time the hash query $m\|r$ was asked. But at this point C has no information about at most q_{sig} totally random elements in L (remember, an element is discarded from L after each signing query). So the probability that $r \in L$ is at most $q_{\mathsf{sig}} 2^{-k_0}$, completing the proof.

Finally, we very briefly sketch the proof of (b). This proof is essentially from [BR93], and is given mainly for the purposes of contrasting it with the proof of (a) above. From the signature forger C, we need to construct an inverter A for $\mathcal{F}$. A picks a random index $\ell \in \{1 \ldots q_{\mathsf{hash}} + 1\}$, hoping that the ℓ-th hash query will be on a new value (not defined in a previous hash query or signature query), and that C will forge a signature based on it (note that if C is successful, such ℓ has to exist). For all hash queries j except for the ℓ-th one, A responds by picking a random x' and returning $f(x')$ (unless the value is already defined, in which case this value is returned). For the ℓ-th one, A responds with its challenge y (unless the hash value is already defined through a previous hash or signing query, in which case A fails). To answer a signing query m_i, $\mathcal{A}$ picks a random r_i and x_i and defines $\mathcal{RO}(m_i\|r_i) = f(x_i)$ (unless it was already defined, in which case it uses the corresponding answer). It then returns $\langle x_i, r_i\rangle$ as the signature of m_i. Finally, C returns a forgery $\langle x, r\rangle$ for some message m with probability ε. If the ℓ-th hash query happens to be exactly $m\|r$, x is the correct preimage of the challenge y. Otherwise, A fails. It is easy to see that A's simulation of the random oracle is perfect and reveals no information about ℓ. Thus, A correctly guesses ℓ with probability $1/(q_{\mathsf{hash}} + 1)$, obtaining a total probability of $\varepsilon/(q_{\mathsf{hash}} + 1)$ of inverting y. □

Notice, the proof of part (a) is indeed identical to the the corresponding proof for RSA [Cor02], except we abstract away all the specifics about RSA. As before, the fact that $k_0 > \log q_{\mathsf{hash}}$ ensures that we can tell apart the hash queries related to q_{sig} signing queries from those maybe related to the forgery. But we see the crucial way the proof uses the claw-freeness of $\mathcal{C}$: the "signing-related" queries get answered with $f(x)$, while the "forging-related" queries get answered with $g(z)$ (where x and z are random to ensure a random answer). In particular, there is no need to guess in advance a *single* "forging-related" query which actually happens to be the one we need: no matter which of these queries will result in the forgery x, one still gets a claw (x, z) such that $f(x) = g(z)$. This should be contrasted with the standard proof of part (b), where we have to guess this query in advance in order to embed our challenge y in the answer. As we show in the next section, the security loss of q_{hash} is optimal for general trapdoor permutations, so the above "guessing" argument cannot be improved.

4 Trapdoor Permutations Cannot Yield Better Security

In this section we explain our "black-box" model and the limitations of proving tighter security results for FDH-like schemes based on general trapdoor permutations. Specifically, our argument will show that the security loss of $\Omega(q_{\mathsf{hash}})$ is inevitable for FDH, PFDH, PSS, showing the tightness of the kind of analysis we used in part (b) of Theorem 1. In order to unify our argument, we consider any signature scheme of the form $\langle f^{-1}(y), pub\rangle$, where $(y, pub) \leftarrow T(m)$ is obtained from the message m using a constant (in our schemes, one or two) number of random oracle calls, possibly some additional randomness, but without utilizing f or f^{-1}. The only thing we require from T in our proof is that for any *distinct* messages $m_1 \ldots m_q$, setting $\langle y_j, pub_j\rangle \leftarrow T(m_j)$ will result in *all distinct* y_j's with all but negligible probability (over the choices of the random oracle, for any q polynomial in the security parameter). In the following, we will call any signature scheme $\mathcal{S}$ (of the above form) utilizing such "collision-resistant" mapping T *legal*. Certainly, FDH, PFDH and PSS are all legal.

Assume now that one claims to have proven a statement of the form: "if $\mathcal{F}$ is (t', ε')-secure, then some particular legal $\mathcal{S}$ is $(t, q_{\mathsf{hash}}, q_{\mathsf{sig}}, \varepsilon)$-secure, for any trapdoor permutation family $\mathcal{F}$." We remark that our lower bound will apply even for proving much weaker $\mathsf{UF\text{-}NMA}$ security (i.e., disallowing the adversary to make any signing queries), so will assume throughout that $q_{\mathsf{sig}} = 0$ and denote $q = q_{\mathsf{hash}}$. A natural proof for such a statement will give a *reduction* R from any adversary C which $(t, q, 0, \varepsilon)$-breaks the unforgeability of $\mathcal{S}$ in the random oracle model, to a forger A which (t', ε')-breaks the one-wayness of f in the standard model. Intuitively, this reduction R (which we sometimes identify with A since the goal of R is to construct A) is *black-box* if it only utilizes the facts that: (1) f is a trapdoor permutation, but the details of f's implementation are unimportant; (2) C $(t, q, 0, \varepsilon)$-breaks $\mathcal{S}$ whenever it is given oracle access to a true random oracle (which has to be "simulated" by R), but the details how C does it are unimportant. In other words, there are *two* objects that a "natural" reduction R (or inverter A) utilizes in a "black-box" manner: the trapdoor permutation f and the forger C. We explain our modeling of each separately.

4.1 Modeling Black-Box Trapdoor Permutation Family $\mathcal{F}$

Following previous work [GT00, GGK02], we model black-box access to a trapdoor permutation $\mathcal{F}$ by three oracles (G, F, F^{-1}), available to all the participants of the system. For simplicity, we will assume that the domain of all our functions is $D = \{0,1\}^k$, both the index i and the trapdoor TK also range over $\{0,1\}^k$, and are in one-to-one correspondence with each other. F is the forward evaluation oracle, which takes the index i of our trapdoor permutation f_i and the input x, and simply returns the value of $f_i(x)$. G takes the trapdoor value TK and returns the index i of the function f_i whose trapdoor is TK (informally, knowing the trapdoor one also knows the function, but not vice versa). Finally, F^{-1} takes the trapdoor TK and the value y and returns $f_i^{-1}(y)$, where $i = G(\mathsf{TK})$. Intuitively, any choice of the oracles G, F, F^{-1} will yield a trapdoor permutation as long as:

(a) $G(\cdot)$ is a permutation over $\{0,1\}^k$.

(b) $F(i, \cdot)$ is a permutation over $\{0,1\}^k$ for every index i.

(c) $F^{-1}(\mathsf{TK}, \cdot)$ is a permutation over $\{0,1\}^k$ for every trapdoor TK.

(d) $F^{-1}(\mathsf{TK}, F(G(\mathsf{TK}), x)) = x$, for any $x, \mathsf{TK} \in \{0,1\}^k$.

(e) For any A, let

$$\mathsf{Adv}_A^{\mathcal{F}}(k) = \Pr[x' = x \mid x, \mathsf{TK} \leftarrow \{0,1\}^k,\ i = G(\mathsf{TK}),\ y = F(i, x),\ x' \leftarrow A^{G,F,F^{-1}}(i, y)] \quad (4)$$

Then we want to require that for any A making polynomial in k number of queries to G, F, F^{-1}, we have $\mathsf{Adv}_A^{\mathcal{F}}(k) = \mathsf{negl}(k)$. For exact security, we say that $\mathcal{F}$ is $(q_G, q_F, q_{F^{-1}}, \varepsilon)$-secure, if $\mathsf{Adv}_A^{\mathcal{F}} \leq \varepsilon$, for any $\mathcal{A}$ making at most q_G queries to G, q_F queries to F and $q_{F^{-1}}$ queries to F^{-1}.[5]

Put differently, we simply rewrote Definition 1, except the efficient computation of f_i and f_i^{-1} (the latter with the trapdoor) are replaced by oracle access to the corresponding oracles, and the notion of "polynomial time" became "polynomial query complexity".[6] In particular, any choice of (G, F, F^{-1}) satisfying conditions (a)-(e) above forms a valid "black-box" trapdoor permutation family. And, therefore, our black-box reduction from forger C to inverter A for $\mathcal{F}$ should work with any such black-box trapdoor permutation. We choose a specific very natural black-box trapdoor permutation for this purpose. G is simply a *random* permutation, and so is $F(i,\cdot)$ (for any i, and independently for different i's). Finally, $F^{-1}(\mathsf{TK}, y)$ is defined in an obvious way so as to satisfy (d) (in particular, it is also a random permutation for any value of TK).

With respect to this family, we can compute the *expected advantage* of any inverter A (taken over the random choice of G, F, F^{-1}). For that, assume that $A(i, y)$ makes q_G queries to G, q_F queries to F and $q_{F^{-1}}$ to F^{-1}. In fact, for future use[7] we will also allow A to make $q_{\tilde{F}^{-1}}$ queries to the new oracle $\tilde{F}^{-1}(i', y') \stackrel{\text{def}}{=} F^{-1}(G^{-1}(i'), y')$, with the obvious restriction that this oracle cannot be called on input (i, y). Without loss of generality, let us assume that A never makes a call to $F^{-1}(\mathsf{TK}', \cdot)$ before first inquiring about $G(\mathsf{TK}')$. In this case, since $F^{-1}(\mathsf{TK}', \cdot)$ is totally random for every TK', there is no need for A to ever call the inverse oracle $F^{-1}(\mathsf{TK}', y')$ more than once: unless $G(\mathsf{TK}') = i$, such call is useless for inverting y, and if the equality holds, calling $F^{-1}(\mathsf{TK}', y)$ immediately inverts y. So we may assume that $q_{F^{-1}} = 0$, and A wins if it ever makes a call $G(\cdot)$ with the "correct" trapdoor $\mathsf{TK} = G^{-1}(i)$, and addition to when it correctly inverts y. Then, naturally extending the definition of $\mathsf{Adv}_A^{\mathcal{F}}$ in Equation (4) to also account for querying $\tilde{F}^{-1}$ on inputs different from (i, y), we show

Lemma 1 $\quad \mathbf{E}[\mathsf{Adv}_A^{\mathcal{F}}(k)] \leq \frac{q_G}{2^k} + \frac{q_F}{2^k - q_{\tilde{F}^{-1}}}$.
In particular, with all but $2^{-\Omega(k)}$ probability, $\mathsf{Adv}_A^{\mathcal{F}}(k) = 2^{-\Omega(k)}$, for any PPQ *adversary A.*

Proof: The first term $q_G/2^k$ is the probability that A calls G on $\mathsf{TK} = G^{-1}(i)$. Assuming that this did not happen, the best strategy of A is to perform $q_{\tilde{F}^{-1}}$ arbitrary queries to $\tilde{F}^{-1}(i, \cdot)$ (with no second input equal to y). This will eliminate $q_{\tilde{F}^{-1}}$ values of x as possible preimages of y, so that there is no need to query $F(i, x)$ for these values of x. Finally, the probability that q_F queries to F will hit one of $(2^k - q_{\tilde{F}^{-1}})$ equally likely possibilities for the needed $F^{-1}(\mathsf{TK}, y)$ is at most the second term. □

Intuitively, the above result says that a truly random family of permutations forms a family of trapdoor permutations with very high probability, since there is no way to invert a random permutation other than by sheer luck.

4.2 Modeling Black-Box Forger C

Let us now turn our attention to the black-box way the reduction R can utilize some signature forger C when constructing the inverter A. (From now on, we identify R and A.) Recall, R is given an index

[5] In the black-box model, the notion of "time" is somewhat less important, since the efficiency conditions for certain functionalities are replaced by having oracle access to these functionalities. So *query complexity* is a more natural complexity measure in this setting.

[6] For this section, we let PPQ stand for "probabilistic polynomial query complexity".

[7] Intuitively, this oracle will correspond to the forgeries returned to A by C.

i for F and a random element y to invert. It is natural to assume that is sets the same public key i for C, and then simply runs $C(i)$ one or more times (possibly occasionally "rewinding" the state of C, but always leaving i as the public key for $\mathcal{S}$). Of course, R has to simulate for C the $q = q_{\mathsf{hash}}$ random oracle queries, since there is no random oracle in the "world of A".[8] Finally, somewhere in the latter simulation, A will utilize the challenge y, so that the forgery returned by C will help A to invert y.

We see that the only thing about C that this kind of reduction is concerned about is the upper bound q on the number of hash queries made by C, and the forged signature returned by C. In other words, from R's perspective, there are $q+1$ rounds of interaction between between R and C: q responses to C's hash queries, followed by a forgery (m, σ, pub) returned by C. Hence, it is very natural to measure the complexity of the reduction (or the inverter A) as the number of rounds of interaction it makes with C.[9] We will call this complexity q_R.

On the other hand, R should succeed in inverting y with the claimed probability ε' for any (q, ε)-*valid* forger C. This means that C is guaranteed to output a forgery to $\mathcal{S}$ with probability ε by making at most q hash queries to R, *provided R answers them in a manner "indistinguishable" for a true random oracle.* On the other hand, it is not important (and the success of R should depend on it) how C managed to obtain the forgery, why it asks some particular hash query, how long it takes C to decide which next hash query to ask, etc. Therefore, in particular, R should succeed even when we give (as we will do in our proof below) to C oracle access to the $\tilde{F}^{-1}$ oracle, which can invert any $\tilde{y}$ that C wishes (this allows C to trivially forge any signature it wants).

Also, we will assume that C can choose randomness in a manner not controllable by R (wlog, C chooses all the randomness it needs at the very beginning of its run). This requires a bit of justification. Indeed, it seems natural to give our reduction the ability to run C with different random tapes. But consider a deterministic C with some particular fixed random tape. In this case, R should not be able to "know" this fixed tape given only oracle access to C, but should still work with this C. But now taking an average over all such C's with a fixed tape, we effectively get that R should work with a forger who chooses its random tape at the beginning, without R "knowing" this choice.

SUMMARY. We are almost done with our modeling, which we now summarize. C is allowed to: (1) choose some arbitrarily large random tape at the beginning (possibly of exponential size since R should work even with computationally unbounded C); (2) have oracle access to $\tilde{F}^{-1}$, allowing it to forge arbitrary signatures; (3) make at most q hash queries to R. On the other hand, provided R answers these queries in a manner indistinguishable from the random oracle, C has to output a forgery (m, x, pub) (i.e., $(m, F(i, x), pub)$ should pass the verification test) with probability at least ε. Similarly, R can: (1) interact with C for at most q_R rounds; (2) call F at most q_F times; (3) call G at most q_G times; (4) rewind C to any prior point in C's run; (5) start a new copy of C where C will choose fresh randomness. Finally, R has to: (1) "properly" answer hash queries of C; (2) invert the challenge y with probability at least ε'. Our objective is to prove an upper bound on ε' as a function of $q_R, q_F, q_G, q, \varepsilon$ in the black-box model outlined above. We do it in the next section.

4.3 Our Bound

Theorem 2 *For any legal signature scheme $\mathcal{S}$ analyzed in the black-box model with general trapdoor permutations, we have*

$$\varepsilon' \leq \left(\frac{q_G}{2^k} + \frac{q_F}{2^k - (2q_R/q)} \right) + 4 \cdot \frac{q_R}{q} \cdot \frac{\varepsilon}{q} \tag{5}$$

[8] Recall, to get a stronger result we assume that C makes no signing queries.

[9] Of course, we should also remember the quantities q_G and q_F, having to do with the black-box access to a trapdoor permutation. We will return to those later.

In particular, for any PPQ *reduction R running C at most a constant number of times (i.e., $q_R = O(q)$), we have $\varepsilon' = O(\varepsilon/q)$, so the reduction loses at least a factor $\Omega(q_{\mathsf{hash}})$ in security.*

Proof: We describe a particular (q, ε)-legal forger C for $\mathcal{S}$. Recall, to sign m, our signature first gets $(y, pub) \leftarrow T(m)$, and then returns (x, pub), where $x = \tilde{F}^{-1}(i, y)$. We also assumed that the transformation T is such that: (1) it calls the random oracle at most a constant number of times (in the proof, we assume this number is 1; the large number will only affect constant 4 in Equation (5)); and (2) when invoked with distinct m_j's, with all but negligible probability all the y_j's are distinct.

First, C chooses a truly random function H from $q/2$-tuples of elements of $\{0,1\}^k$ to a random index $\ell \in \{1 \dots q/2\}$. This function will control the index of a forgery that C will compute later. Next, C uses q hash query calls to obtain $(y_j, pub_j) \leftarrow T(m_j)$ (if needed, using some additional randomness prepared separately from the function H) for q arbitrary, but distinct and fixed messages $m_1 \dots m_q$. If any of the answers y_j are the same, C rejects, since we assumed that with a true random oracle such collision does not happen with all but negligible probability (i.e., that $\mathcal{S}$ is legal). Otherwise, if all of them are distinct, C computes $\ell = H(y_1 \dots y_{q/2})$, and then, with probability ε, outputs a forged signature $\left\langle \tilde{F}^{-1}(i, y_\ell), pub_\ell \right\rangle$ of m_ℓ.

Clearly, this C is (q, ε)-legal, so our reduction R should be able to invert y with this C, with probability at least ε'. Before arguing that this ε' must satisfy Equation (5), we observe the following. The number of times R has to interact with C between any two successive *distinct* forgeries is at least $q/2$. Indeed, unless any of the values $y_1 \dots y_{q/2}$ change, C will return exactly the same forgery to R, since the function H is fixed by now. So R has to either start a new copy of C and wait for q steps, or rewind one of the current copies of C to at least some query $j \leq q/2$, somehow change the value y_j by returning a different random oracle answer, and then run C for another $q - j \geq q/2$ steps. In particular, R never sees more than $N \stackrel{\text{def}}{=} 2q_R/q$ different forgeries (in fact, the actual number is roughly εN, but this is not important).

Let us now estimate the success probability of R. Let E denote the event that C ever returns the inverse of y to R, i.e. it happens that $y_\ell = y$ for one of the forgeries returned by C. Clearly, $\Pr(R \text{ succeeds}) \leq \Pr(E) + \Pr(R \text{ succeeds} \mid \overline{E}) \stackrel{\text{def}}{=} p_1 + p_2$. We start with estimating p_2. Notice, since E did not happen, we get that C never called $\tilde{F}^{-1}(i, y)$. Thus, combining R and C into one "super-inverter" A', we get that A' made q_G calls to G, q_F calls to F and at most N calls to $\tilde{F}^{-1}$ on inputs different from (i, y). By Lemma 1, we get that $p_2 \leq \frac{q_G}{2^k} + \frac{q_F}{2^k - N}$.

As for p_1, recall that the number of different forgeries that C can return is at most the number of times it evaluates H on a different $q/2$-tuple of values, which in turn is at most N, as we just argued. Indeed, the input to H predetermines the forgery that C can return, so these inputs should be different for different forgeries, and it takes R at least $q/2$ steps to "change" the input to H. Take any one of these (at most) N times with the input to H being $y_1 \dots y_{q/2}$. Since C will never proceed with a forgery if at least two of the y_j's are the same, at most one of the first $q/2$ values of y_j can be equal to y. Since H is truly random function, the probability that it will return ℓ such that $y_\ell = y$ is at most $2/q$. Moreover, even if this event happened, the probability it inverts this y is ε, giving an overall probability of at most $2\varepsilon/q$ of inverting y, each time C might output a new forgery. By the union bound, the overall probability of the event E is $p_1 \leq N \cdot 2\varepsilon/q$. Combining the bounds we got for p_1 and p_2, we get Equation (5). □

Intuitively, the argument above said that the reduction must guess the "relevant" hash query where it can give the answer dependent on the challenge y. And it can do it for only one query since the signature $\mathcal{S}$ is legal and there is no "structure" to a random trapdoor permutation which would allow to use random but "related" values of y for the other hash queries. So this guess is correct with probability only $1/q$. Finally, we remark that the term q_R/q in Equation (5) can be roughly interpreted as the number of times our reduction ran C. Not surprisingly, running C from scratch q_R/q times will improve the chances of success by a factor proportional to q_R/q, which is indeed the behavior we see in Equation (5). For $q_R/q = O(1)$, however, we see that we lose the factor $\Omega(q)$.

5 Some Constructions of Claw-free Permutations

Since we know so few number-theoretic constructions of trapdoor permutations (essentially RSA, Rabin and Paillier [Pai99]), we know very few constructions of claw-free permutations as well. Luckily, every current trapdoor permutation *we know* in fact yields some natural claw-free permutation. On the other hand, we mentioned that this implication from trapdoor to claw-free permutations is very unlikely to hold in general. Therefore, in this section we give several *general conditions* on trapdoor permutations (enjoyed by currently known trapdoor permutations), which suffice to imply the existence of claw-free permutations (in fact, via very efficient constructions). These conditions can be viewed as narrowing the gap between the general claw-free permutations and the very specific ones based on trapdoor permutations like RSA. We also point out at the end of this section that that not all known claw-free permutations actually follow the general constructions we present below.

Claw-Free Permutations from Homomorphic Trapdoor Permutations. This is the most natural generalization of the RSA-based construction presented in Section 2.1. Assume we have a family $\mathcal{F}$ of trapdoor permutations with two group operations $+$ and $\odot$ so that each $f \in \mathcal{F}$ is homomorphic with respect to these operations: $f(a+b) = f(a) \odot f(b)$. We can construct the following claw-free permutation family $\mathcal{C}$ out of $\mathcal{F}$. $\mathsf{CF\text{-}Gen}(1^k)$ runs $(f, f^{-1}) \leftarrow \mathsf{TC\text{-}Gen}(1^k)$, also picks a random $y \in D$, sets $g_y(b) = y \odot f(b)$, and outputs (f, f^{-1}, g_y). Now finding a claw (a, b) implies that $f(a) = y \odot f(b)$ which means that $f(a-b) = y$, which means that $a - b = f^{-1}(y)$, so we manage to invert a trapdoor permutation f on a random point y.

Claw-Free Permutations from Random-Self-Reducible Trapdoor Permutations. This is a further generalization of the previous construction. Assume, there are efficient functions I and O which satisfy the following conditions. For *any* output value $y \in D$, picking a random b and applying $O(y, b)$ results in a random point $z \in D$ (notice, $O(y, \cdot)$ does not have to be a permutation or to be invertible). Then, if one finds out the value $a = f^{-1}(z)$, applying $I(y, a, b)$ will result in finding the correct value $x = f^{-1}(y)$. So one effectively reduces the worst-case task of inverting y to an average-case task of inverting a *random* z. We say that such f is *random-self-reducible* (RSR) if O and I satisfying the above conditions exist. Notice, homomorphic f is RSR via $z = O(y, b) \stackrel{\text{def}}{=} y \odot f(b)$ (which is actually $f(x+b)$). Then $a = f^{-1}(z) = x + b$, so we can define $I(y, a, b) \stackrel{\text{def}}{=} a - b$.

We can construct the following claw-free permutation family $\mathcal{C}$ out of any RSR trapdoor permutation family $\mathcal{F}$. $\mathsf{CF\text{-}Gen}(1^k)$ runs $(f, f^{-1}) \leftarrow \mathsf{TC\text{-}Gen}(1^k)$, also picks a random $y \in D$, sets $g_y(b) = O(y, b)$, and outputs (f, f^{-1}, g_y). Now, finding a claw (a, b) implies that $f(a) = O(y, b)$ which means that $I(y, a, b) = f^{-1}(y) = x$. Thus, we inverted a trapdoor permutation f on a random point y.

An ad hoc claw-free permutation. Any of the above constructions can be applied to trapdoor permutations like RSA, Rabin and Paillier. However, we know of some ad hoc constructions of claw-free permutations which do not follow the above methodology. One such example (based on factoring Blum-Willams integers) is the original claw-free permutation family of [GMR88]. Here $n = pq$, where $p \equiv 3 \bmod 4$, $q \equiv 7 \bmod 8$, and $QR(n)$ stands for the group of quadratic residues modulo n. Then we set our domain $D = QR(n)$, and $f(a) = a^2 \bmod n$, $g(b) = 4b^2 \bmod n$. If $f(a) = g(b)$ for $a, b \in QR(n)$, then $(a - 2b)$ is divisible by p or q, but not n, which allows one to factor n.

References

[ACM89] *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, Seattle, Washington, 15–17 May 1989.

[BM88] Mihir Bellare and Silvio Micali. How to sign given any trapdoor function. In Goldwasser [Gol88], pages 200–215.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communication Security*, pages 62–73, November 1993. Revised version appears in `http://www-cse.ucsd.edu/users/mihir/papers/crypto-papers.html`.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli Maurer, editor, *Advances in Cryptology—EUROCRYPT 96*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416. Springer-Verlag, 12–16 May 1996. Revised version appears in `http://www-cse.ucsd.edu/users/mihir/papers/crypto-papers.html`.

[Cor00] Jean-Sébastian Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *Advances in Cryptology—CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235. Springer-Verlag, 20–24 August 2000.

[Cor02] Jean-Sébastian Coron. Optimal security proofs for PSS and other signature schemes. In Lars Knudsen, editor, *Advances in Cryptology—EUROCRYPT 2002*, Lecture Notes in Computer Science, pages 272–287. Springer-Verlag, 28 April–2 May 2002.

[Dam87] Ivan Damgård. Collision-free hash functions and public-key signature schemes. In David Chaum and Wyn L. Price, editors, *Advances in Cryptology—EUROCRYPT 87*, volume 304 of *Lecture Notes in Computer Science*. Springer-Verlag, 1988, 13–15 April 1987.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology—CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987, 11–15 August 1986.

[GGK02] Rosario Gennaro, Yael Gertner, and Jonathan Katz. Bounds on the efficiency of encryption and digital signatures. Technical Report 2002-22, DIMACS: Center for Discrete Mathematics and Theoretical Computer Science, 2002. Available from `http://dimacs.rutgers.edu/TechnicalReports/2002.html`.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GMR01] Yael Gertner, Tal Malkin, and Omer Reingold. On the impossibility of basing trapdoor functions on trapdoor predicates. In *42nd Annual Symposium on Foundations of Computer Science*, Las Vegas, Nevada, October 2001. IEEE.

[Gol88] Shafi Goldwasser, editor. *Advances in Cryptology—CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990, 21–25 August 1988.

[GQ88] Louis Claude Guillou and Jean-Jacques Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In Goldwasser [Gol88], pages 216–231.

[GT00] Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *41st Annual Symposium on Foundations of Computer Science*, Redondo Beach, California, November 2000. IEEE.

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In ACM [ACM89], pages 44–61.

[KR00] Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *Network and Distributed System Security Symposium*, pages 143–154. The Internet Society, 2000.

[KST99] Jeong Han Kim, Daniel R. Simon, and Prasad Tetali. Limits on the efficiency of one-way permutation-based hash functions. In *40th Annual Symposium on Foundations of Computer Science*, New York, October 1999. IEEE.

[Mic94] Silvio Micali. A secure and efficient digital signature algorithm. Technical Report MIT/LCS/TM-501, Massachusetts Institute of Technology, Cambridge, MA, March 1994.

[MR02] Silvio Micali and Leonid Reyzin. Improving the exact security of digital signature schemes. *Journal of Cryptology*, 15:1–18, 2002.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In ACM [ACM89], pages 33–43.

[OS90] Heidroon Ong and Claus P. Schnorr. Fast signature generation with a Fiat Shamir-like scheme. In I. B. Damgård, editor, *Advances in Cryptology—EUROCRYPT 90*, volume 473 of *Lecture Notes in Computer Science*, pages 432–440. Springer-Verlag, 1991, 21–24 May 1990.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology—EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*. Springer-Verlag, 2–6 May 1999.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 387–394, Baltimore, Maryland, 14–16 May 1990.

[Sim98] Daniel R. Simon. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions. In Kaisa Nyberg, editor, *Advances in Cryptology—EUROCRYPT 98*, volume 1403 of *Lecture Notes in Computer Science*. Springer-Verlag, May 31–June 4 1998.