Efficiency Improvements for Signature Schemes with Tight Security Reductions

Jonathan Katz
Dept. of Computer Science
University of Maryland
College Park, MD
jkatz@cs.umd.edu

Nan Wang*
Dept. of Computer Science
University of Maryland
College Park, MD
nwang@cs.umd.edu

ABSTRACT

Much recent work has focused on constructing *efficient* digital signature schemes whose security is *tightly* related to the hardness of some underlying cryptographic assumption. With this motivation in mind, we show here two approaches which improve both the computational efficiency and signature length of some recently-proposed schemes:

Diffie-Hellman signatures. Goh and Jarecki [18] recently analyzed a signature scheme which has a tight security reduction to the computational Diffie-Hellman problem. Unfortunately, their scheme is less efficient in both computation and bandwidth than previous schemes relying on the (related) discrete logarithm assumption. We present a modification of their scheme in which signing is 33% more efficient and signatures are 75% shorter; the security of this scheme is tightly related to the decisional Diffie-Hellman problem.

PSS. The probabilistic signature scheme (PSS) designed by Bellare and Rogaway [3] uses a random salt to enable a tight security reduction to, e.g., the RSA problem. Coron [12] subsequently showed that a shorter random salt can be used without impacting the security of the scheme. We show a variant of PSS which avoids the random salt *altogether* yet has an equally-tight security reduction. This furthermore yields a version of PSS-R (PSS with message recovery) with *optimal* message length. Our technique may also be used to improve the efficiency of a number of other schemes.

Categories and Subject Descriptors

E.3 [**Data Encryption**]: Public-Key Cryptosystems

General Terms

Algorithms, Security, Theory

*Research supported in part by National Science Foundation grant CCR-0208005.

CCS'03, October 27–30, 2003, Washington, DC, USA.

Keywords

Digital Signatures

1. INTRODUCTION

Beginning in the early 1980's [19], the science of cryptography has focused on constructing signature schemes that can be rigorously proven secure based on specific computational assumptions. A proof of security for a given construction generally proceeds by a reduction showing how an adversary "breaking" the scheme in polynomial time can be used to "solve" in polynomial time some underlying problem assumed to be difficult (e.g., RSA). Classically, results of this sort have been *asymptotic*; i.e., the security reduction only guarantees that, as the security parameter (e.g., key length) increases, no poly-time adversary can forge signatures with "sufficiently high" probability. As first emphasized by Bellare and Rogaway [3], however, such results say little about the security of a scheme *in practice* for a particular choice of key size and against adversaries investing a specific amount of computational effort. For practical considerations it is critical to focus on *concrete security reductions* which give explicit bounds on an adversary's success probability as a function of their expended resources.

We provide a simplified example which we hope will provide further motivation (see [3, 11, 12, 18] for further discussion). Assume a signature scheme (relying on some appropriate computational assumption) which an adversary expending 1 year of computational effort can "break" with probability at most $a \cdot 2^{-b \cdot \kappa}$, where κ is the key size and a, b are constants. Under an asymptotic definition of security, this scheme is secure. Yet, in practice, we do not know what key size to choose unless a and b are known! Furthermore, the values of a, b are crucial for determining the efficiency of the scheme: for a desired security level (i.e., probability of adversary forgery) of 2^{-32} following 1 year of effort, having $a \approx 1$ and $b \approx 1/10$ means that κ should be roughly 320; on the other hand, if $a \approx 2^{32}$ and $b \approx 1/20$ then we require $\kappa \approx 1280$ with a concomitant decrease in efficiency to achieve the same level of security.

The above discussion illustrates that comparisons of the efficiency of two signature schemes must take into account the relative security each scheme offers; alternately, such a comparison must take into account the efficiency of the security reduction (recall, this shows how an adversary "breaking" the signature scheme can be used to "break" some problem assumed to be hard). In some sense, the most efficient

reduction we can hope for is one in which an adversary who "breaks" a signature scheme with probability ε in time t can be used to "break" the underlying computational problem with probability $\varepsilon' \approx \varepsilon$ in time $t' \approx t$; a reduction of this sort is called *tight*. A scheme with a non-tight reduction will necessarily require larger key sizes to provide the same security as a scheme with a tight security reduction. Indeed, it is often the case that obtaining a reasonable level of security from a scheme with a non-tight reduction requires using a key length which is completely impractical!

These observations have sparked a significant amount of research aimed at developing efficient signature schemes with security reductions as tight as possible. As an example, consider "hash-and-sign" signatures based on trapdoor permutations (e.g., RSA). Let ε' be the probability of inverting a *specified* trapdoor permutation (e.g., RSA with 1024-bit moduli) in some time t'; in the discussion which follows, one may take t' to be 1 year and $\varepsilon' \approx 2^{-60}$ for concreteness but the argument is generally applicable. The full-domain-hash (FDH) signature scheme [1, 3] bounds the probability of forgery by $\varepsilon \approx (q_s + q_h)\varepsilon'$ for any adversary running in time $t \approx t'$, where q_s (resp., q_h) is the number of signatures (resp., hash function evaluations) obtained by the adversary. Since signatures can only be obtained from the legitimate signer while the adversary can evaluate the hash function on its own, it has been suggested [3, 11, 12] to use $q_s \approx 2^{30}$ and $q_h \approx 2^{60}$. But if we take $\varepsilon' \approx 2^{-60}$ then the security guarantee provided by the signature scheme is meaningless (since $\varepsilon \approx 1$)! We stress that it is meaningless only for specific trapdoor permutations with $\varepsilon' \approx 2^{-60}$; by choosing a different trapdoor permutation (say, RSA with 2048-bit moduli) with smaller ε', meaningful results can be obtained.

This again illustrates the effect of concrete security on efficiency: In the above example, one must use a trapdoor permutation with very low probability of inversion (e.g., by using longer key sizes) in order to obtain any security at all.

Coron [11] has subsequently shown how to improve the security reduction for FDH for the *specific* case when RSA is used as the trapdoor permutation[1]. Coron's results bound the probability of forgery in this case by $\varepsilon \approx q_s\varepsilon'$. Returning to our above example, choosing our RSA key length so that $\varepsilon' \approx 2^{-60}$ now gives a reasonable guarantee of security (i.e., $\varepsilon \approx 2^{-30}$).

To further improve the tightness of the security reduction for "hash-and-sign"-type signatures, the probabilistic signature scheme (PSS) was introduced [3] and shown to achieve tight security reduction $\varepsilon \approx \varepsilon'$ when RSA is used as the trapdoor permutation (although footnote 1 applies here as well). The key feature distinguishing PSS from FDH is that in the former there are multiple valid signatures corresponding to any given message. This is used in an essential way in the proof of security; indeed, it has been shown [12] that "hash-and-sign" signature schemes without this property cannot have a tight security reduction.

To ensure multiple valid signatures for a message, PSS uses a "salt" r which is randomly generated and hashed along with the message each time a signature is produced. The original work [3] required $|r| = 180$ (for values of q_s, q_h as above) to achieve a tight security reduction. Coron later observed [12] that the length of the salt could be reduced to $|r| = 30$ while obtaining essentially the same security bound (Coron also shows [12] that this value of $|r|$ is optimal for PSS). Reducing the length of r is important since it reduces the randomness used. More importantly, when extending PSS to provide message recovery, the length of the recoverable messages increases as the length of r decreases.

The above discussion concerns what may be termed "FDH-like" schemes. In other related work, Micali and Reyzin [23] improve the exact security of some signature schemes derived using the Fiat-Shamir heuristic [16]. More recently, Goh and Jarecki [18] analyzed a signature scheme with a tight security reduction to the computational Diffie-Hellman problem; interestingly, this scheme is the first discrete-log based scheme with tight security (in particular, the scheme avoids the "forking lemma" of [24]). We stress that the goal of achieving tight security reductions is not limited to signature schemes; the issue is critical for public-key encryption (see, for example, [28, Section 1.3]) and there has been much research in this area as well (e.g., [2, 5, 17, 27, 21]).

The above-mentioned schemes are all analyzed in the *random oracle model* [16, 1], which is also used here. It is crucial to note that a proof of security in the random oracle model does not guarantee security when this oracle is instantiated by any particular cryptographic hash function [9]. However, a proof in this model does indicate that there are no "inherent" weaknesses in the scheme itself; thus, a scheme with a proof of security in the random oracle model is clearly preferable to a scheme with no proof of security at all.

1.1 Summary of Results

Motivated by the above line of research, we focus on *efficient* signature schemes with *tight* security reductions, and improve the efficiency (both in terms of computation and signature length) of a number of recently-proposed schemes.

Diffie-Hellman signatures. Various discrete-log based signature schemes, providing alternatives to RSA-based signatures, are known [14, 25, 15, 8]. Proofs of security for these schemes — when available — are related to the hardness of computing discrete logarithms in some underlying group. Interestingly, Goh and Jarecki [18] recently noted that none of these schemes has a *tight* security reduction to the discrete logarithm problem; this motivated their analysis of a signature scheme proposed previously [10] whose security they show is tightly related to the hardness of the computational Diffie-Hellman (CDH) problem in some underlying group. They further argue [18] that basing security on this, possibly stronger, assumption is not a severe drawback since for many cryptographic groups used in practice the CDH problem is as hard as the discrete logarithm problem (using current techniques) [4, 22]. Unfortunately, the scheme they analyze is less efficient than previous discrete-log based schemes: signing requires 3 modular exponentiations, and signatures are roughly 1400 bits long for the key sizes and cryptographic groups they recommend.

We show here a simple modification of their scheme which results in much better efficiency: signing requires only 2 exponentiations (both of which may be computed "off-line" before the message is known) and signatures are only 320 bits long. The security of our scheme is tightly related to the hardness of the *decisional* Diffie-Hellman (DDH) problem rather than the CDH problem. Although the DDH problem is possibly easier than the CDH problem in general, we note (as in [18]) that these problems are equally hard for many commonly-used groups *as far as current techniques*

[1]Actually, this result can be generalized [11, 13] but this is unimportant for the present discussion.

are concerned. We discuss this further in Section 2.2.

PSS. Recall that PSS [3] is a probabilistic variant of FDH which introduces a random salt r to achieve a tight security reduction. (The general technique of using a random salt to achieve a tight(er) security reduction is applicable to other schemes also; for example, the technique may be used to improve the exact security of the "short signatures" of [7] and is explicitly used to obtain tight security in [18].) As noted by Coron [12], the original analysis of Bellare and Rogaway requires $|r| \geq 2 \cdot \log_2 q_h + \log_2 1/\varepsilon'$ (where the notation is as in the previous section); taking $q_h \approx 2^{60}$ and $\varepsilon' \approx 2^{-60}$ gives a salt length $|r|$ of 180 bits. Refining the analysis, Coron showed that a tight security reduction is obtained even for $|r| \geq \log_2 q_s$; this leads to a substantial improvement since, in practice, we have $q_s \approx 2^{30}$.

Here, we show how to avoid the random salt *altogether* while still obtaining a tight proof of security. This does not contradict the result of Coron [12, Theorem 5] that a tight security reduction is impossible for "FDH-like" schemes with unique signatures since our scheme is constructed such that every message has exactly *two* signatures. However, although a given message has more than one valid signature, only one of these will ever be produced by the legitimate signer *even if the message is signed multiple times*. We accomplish this via a deterministic signing algorithm that does not require the signer to maintain any state, It is interesting that this subtle change is sufficient to enable a tight proof of security. Our proposed modification easily extends to eliminate the need for a random salt in, e.g., [7, 18] as well.

For schemes in which the random salt is included with the signature, our technique yields signatures of shorter length (equivalently, for signatures of the same length we obtain better security). Thus, we may modify the scheme of [18] to save 111 bits while still basing security on the CDH assumption, or we can modify the "short signatures" of [7] to obtain signatures of the same length but with improved security. Furthermore, in the case of PSS-R (i.e., PSS with message recovery) a shorter salt translates to the ability to sign longer messages; thus, with our technique PSS-R can be used for messages 30 bits longer than in [12] (under the same assumptions, and with the same level of security). In fact, combining our technique with ideas of Granboulan [20] gives a version of PSS-R with tight security which we show is *optimal* in terms of allowable message length.

2. DEFINITIONS

We review definitions of signature schemes, the decisional Diffie-Hellman assumption, and claw-free (trapdoor) permutations. Since we analyze our schemes in terms of their concrete security, none of our definitions explicitly refers to any "security parameter". We stress, however, that all our results imply security in the asymptotic sense as well.

2.1 Signature Schemes

We give a functional definition of both general signature schemes as well as those supporting message recovery; this is followed by a definition of security. In the following, H refers to a hash function to which the algorithms are given oracle access; this hash function will be treated as a random oracle in the analysis. In practice, as suggested in [1], H will be instantiated by a cryptographic hash function mapping the appropriate domain to the appropriate range.

DEFINITION 1. *A signature scheme is a tuple of probabilistic algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *over a message space* $\mathcal{M}$ *such that:*

- *The key-generation algorithm* Gen *outputs a public key* PK *and a secret key* SK.
- *The signing algorithm* Sign *takes as input secret key* SK *and message* $m \in \mathcal{M}$ *and returns signature* σ.
- *If message recovery is not supported, the verification algorithm* Vrfy *takes as input a public key* PK, *a message* $m \in \mathcal{M}$, *and a signature* σ *and returns* accept *or* reject.
- *If message recovery is supported, verification algorithm* Vrfy *takes as input public key* PK *and signature* σ *and returns either a message* $m \in \mathcal{M}$ *or* reject.

We make the standard correctness requirement, given here for schemes supporting message recovery (the other case is analogous): for all SK, PK *output by* Gen *and all* $m \in \mathcal{M}$ *we have* $\mathsf{Vrfy}_{PK}(\mathsf{Sign}_{SK}(m)) = m$.

We now give a definition of strong unforgeability under adaptive chosen-message attacks (cf. [19]); "strong" here means that the adversary cannot even generate a new signature for a previously-signed message. We let Ω denote the space from which the random oracle H is selected.

DEFINITION 2. *Let* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *be a signature scheme which supports message recovery. An adversarial forging algorithm* $\mathcal{F}$ *is said to* $(t, q_h, q_s, \varepsilon)$*-break this scheme if* $\mathcal{F}$ *runs in time at most* t, *makes at most* q_h *hash queries and at most* q_s *signing queries, and furthermore*

$$\Pr[(PK, SK) \leftarrow \mathsf{Gen}; H \leftarrow \Omega; \sigma \leftarrow \mathcal{F}^{\mathsf{Sign}_{SK}(\cdot), H(\cdot)}(PK) : \sigma \notin \Sigma \wedge \mathsf{Vrfy}_{PK}(\sigma) \neq \mathsf{reject}] \geq \varepsilon,$$

where Σ *is the set of signatures received from* $\mathsf{Sign}_{SK}(\cdot)$.

If signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *does not support message recovery, the definition is as above except that we are interested in the probability that* $\mathcal{F}$ *outputs a pair* (m, σ) *such that* $\mathsf{Vrfy}_{PK}(m, \sigma) = \mathsf{accept}$ *but* σ *was never the response of a query* $\mathsf{Sign}_{SK}(m)$.

In either case, a signature scheme is $(t, q_h, q_s, \varepsilon)$*-secure if no forger can* $(t, q_h, q_s, \varepsilon)$*-break it.*

2.2 The Decisional Diffie-Hellman Problem

Let $\mathbb{G}$ be a finite, cyclic group of prime order q in which the group operation is represented multiplicatively; furthermore, let g be a generator of $\mathbb{G}$. The decisional Diffie-Hellman (DDH) problem may be described, informally, as the problem of distinguishing between tuples of the form (g, g^x, g^y, g^z) for random $x, y, z \in \mathbb{Z}_q$ (these are denoted "random tuples") and tuples of the form (g, g^x, g^y, g^{xy}) for random $x, y \in \mathbb{Z}_q$ (these are denoted "Diffie-Hellman tuples"). The following definition makes this more concrete.

DEFINITION 3. *A distinguishing algorithm* $\mathcal{D}$ *is said to* (t, ε)*-break DDH in group* $\mathbb{G}$ *if* $\mathcal{D}$ *runs in time at most* t *and furthermore*

$$|\Pr[x, y, z \leftarrow \mathbb{Z}_q : \mathcal{D}(g, g^x, g^y, g^z) = 1] - \Pr[x, y \leftarrow \mathbb{Z}_q : \mathcal{D}(g, g^x, g^y, g^{xy}) = 1]| \geq \varepsilon.$$

We say that $\mathbb{G}$ *is a* (t, ε)*-DDH group if no algorithm* (t, ε)*-breaks DDH in* $\mathbb{G}$.

Hardness of the DDH problem in $\mathbb{G}$ implies hardness of the computational Diffie-Hellman (CDH) problem as well as hardness of the discrete logarithm problem in $\mathbb{G}$. The converse is not, in general, true; there are groups in which the DDH problem is known to be easy yet the CDH or discrete logarithm problems seem to be hard. However, for a variety of groups of interest "*the best known algorithm for DDH is a full discrete log algorithm*" [4] (see also the discussion by Maurer and Wolf [22]). These include, among others, the commonly-used group $\mathbb{G}$ of quadratic residues modulo p, where $p = \alpha q + 1$ is prime. Additionally, Shoup [26] has shown that the DDH problem is as hard as the discrete logarithm problem for *generic* algorithms (i.e., those that do not use the underlying group structure). For more details, the reader is referred to two excellent surveys [4, 22].

2.3 Claw-Free Permutations

As briefly touched upon in the Introduction, FDH was originally proven secure when an arbitrary trapdoor permutation is used [1]. On the other hand, subsequent work on FDH [11] and PSS [3, 12] has focused on specific classes of trapdoor permutations such as the RSA and Rabin permutations. Dodis and Reyzin have since noted [13] that the unifying feature of these subsequent analyses is their reliance on *trapdoor permutations induced by claw-free permutations* (of which the RSA and Rabin permutations are two examples). For this reason, we introduce this notion now.

The notion of claw-free permutations generalizes that of trapdoor permutations. Informally, a pair of claw-free permutations (f_0, f_1) has the property that f_0 and f_1 are each individually trapdoor permutations over the same domain; furthermore, this pair has the additional property that, without one of the associated trapdoors, it is hard to find a "claw" (namely, x_0 and x_1 such that $f_0(x_0) = f_1(x_1)$). We give a formal definition here.

DEFINITION 4. *A family of claw-free permutations is a tuple of* PPT *algorithms* $(\mathsf{cf\text{-}Gen}, \mathsf{F}, \mathsf{G}, \mathsf{F}^{-1}, \mathsf{G}^{-1})$ *such that:*

- $\mathsf{cf\text{-}Gen}$ *outputs a random index i and a trapdoor* td.
- $\mathsf{F}(i,\cdot)$ *and* $\mathsf{G}(i,\cdot)$ *are both permutations over the same domain, denoted D_i. Furthermore, there is an efficient sampling algorithm which, on input i, outputs a uniformly-distributed element in D_i.*
- *If* (i, td) *was output by* $\mathsf{cf\text{-}Gen}$*, then* $\mathsf{F}^{-1}(\mathsf{td},\cdot)$ *(resp.,* $\mathsf{G}^{-1}(\mathsf{td},\cdot)$*) is the inverse of* $\mathsf{F}(i,\cdot)$ *(resp.,* $\mathsf{G}(i,\cdot)$*).*

Algorithm A is said to (t,ε)-break a family of claw-free permutations if A runs in time at most t and furthermore A outputs a "claw" with probability greater than ε. More formally,

$$\Pr[(i,\mathsf{td}) \leftarrow \mathsf{cf\text{-}Gen}; (x_0,x_1) \leftarrow A(i) : \mathsf{F}(i,x_0) = \mathsf{G}(i,x_1)] \geq \varepsilon.$$

A given claw-free permutation is (t,ε)-secure if no algorithm can (t,ε)-break it.

For notational convenience, we will write $f(\cdot)$ instead of $\mathsf{F}(i,\cdot)$ (and similarly for g) when the index i is clear from the context. Similarly, we write $f^{-1}(\cdot)$ instead of $\mathsf{F}^{-1}(\mathsf{td},\cdot)$ (and similarly for g) with the implicit understanding that this inverse cannot be efficiently computed without knowledge of the trapdoor. We also speak of "claw-free permutations" rather than claw-free permutations families.

Note that the tuple $(\mathsf{cf\text{-}Gen}, \mathsf{F}, \mathsf{F}^{-1})$ is a trapdoor permutation (and similarly for $(\mathsf{cf\text{-}Gen}, \mathsf{G}, \mathsf{G}^{-1})$). In this case, following [13], we will say that this trapdoor permutation is induced by a claw-free permutation.

From a complexity-theoretic point of view, the existence of claw-free permutations represents a stronger assumption than the existence of trapdoor permutations. On the other hand, all known examples of trapdoor permutations can be viewed as being induced by a family of claw-free permutations (cf. [13]). In the case of RSA, for example, given a modulus N, an exponent e, and a random value $y \in \mathbb{Z}_N^*$ we may define $f(x) \stackrel{\text{def}}{=} x^e \bmod N$ and $g(x) \stackrel{\text{def}}{=} x^e y \bmod N$; note that finding x_0, x_1 such that $f(x_0) = g(x_1)$ is equivalent to finding a value (x_0/x_1) which is an e^{th} root of y (and is therefore infeasible under the assumption that RSA is a trapdoor permutation). It should also be stressed that, in the above example, RSA yields a (t,ε)-secure claw-free permutation iff RSA itself is a $(O(t),\varepsilon)$-secure trapdoor permutation; in other words, there is essentially no loss of security when translating the problem from one domain to the other.

3. SIGNATURE SCHEME BASED ON DDH

We begin by reviewing the signature scheme analyzed by Goh and Jarecki [18], whose security is tightly related to the CDH problem in some cyclic group $\mathbb{G}$. The basic idea of the scheme is as follows: the public key contains $g, h \in \mathbb{G}$ and the private key consists of $x = \log_g h$. To sign a message m, the message is first mapped (using a random oracle) to a value $z = H(m) \in \mathbb{G}$. The signer then computes z^x and a non-interactive zero-knowledge proof π (using the Fiat-Shamir heuristic [16]) that (g, h, z, z^x) forms a Diffie-Hellman tuple; the signature contains z^x and π. The above description (which slightly simplifies [18]) corresponds roughly to an "FDH-like" signature scheme; their actual scheme uses a 111-bit random salt (à la PSS and probabilistic-FDH [12]) to achieve tight security.[2]

The Goh-Jarecki scheme is very similar in spirit to the "short signatures" previously introduced by Boneh, Lynn, and Shacham [7]. In the latter case, however, the DDH problem is *easy* in the underlying group and therefore the non-interactive proof π is unnecessary. Because of this requirement that DDH be easy, the techniques of this section do not apply to the Boneh-Lynn-Shacham scheme (however, our techniques of Section 4 do apply; see Section 4.4).

Compared to previous signature schemes based on the (weaker) discrete logarithm assumption, the Goh-Jarecki scheme has some notable drawbacks. The signer must compute both z^x and π (resulting in a total of 3 exponentiations), and the signature must include both of these values. For many groups of interest, this leads to very long signatures; for example, in the running example of [18] group elements (e.g., z^x) are roughly 1000-bits long while the entire proof π is 320-bits long.

We present here a way to improve the efficiency and signature length of the above scheme, while maintaining a tight security reduction; the difference is that security of our scheme is based on the DDH problem instead of the CDH problem. In our scheme, the public key consists of a Diffie-Hellman tuple (g, h, y_1, y_2) while the secret key is the

[2]Interestingly, our techniques of Section 4 may be used to eliminate this random salt while preserving the tight security reduction to the CDH problem; see Section 4.4.

value x such that $g^x = y_1$ and $h^x = y_2$. Now, the signature for a message m simply consists of a non-interactive zero-knowledge proof (using the Fiat-Shamir heuristic [16]) that (g, h, y_1, y_2) indeed forms a Diffie-Hellman tuple. Forgery will now be infeasible because the "challenge" in the Fiat-Shamir heuristic depends on m; because the proof is zero-knowledge (informally), viewing multiple proofs of the statement will not help an adversary in constructing a new proof of the statement for a different message.

3.1 Details and Proof of Security

In what follows, we assume a finite, cyclic group $\mathbb{G}$ of prime order q; this group may be fixed or a description of $\mathbb{G}$ may be included with the signer's public key. We let $H : \{0,1\}^* \to \mathbb{Z}_q$ be a hash function which will be modeled as a random oracle.

$\underline{\mathsf{Gen}}$ chooses random generators $g, h \in \mathbb{G}$ and a random value $x \leftarrow \mathbb{Z}_q$. It then computes $y_1 = g^x$ and $y_2 = h^x$. The public key is $PK = (g, h, y_1, y_2)$ and the secret key is x.

$\underline{\mathsf{Sign}_{SK}(m)}$ (where $m \in \{0,1\}^*$) executes a non-interactive zero-knowledge proof that the public key (g, h, y_1, y_2) forms a Diffie-Hellman tuple; this can be done efficiently given the witness x (i.e., the secret key). In more detail: the algorithm chooses random $r \in \mathbb{Z}_q$ and computes $A = g^r, B = h^r$, and "challenge" $c = H(PK, A, B, m)$. Finally, it computes $s = cx + r$ and outputs the signature (c, s).

$\underline{\mathsf{Vrfy}_{PK}(m, \sigma)}$ parses PK as (g, h, y_1, y_2) and σ as (c, s). It then computes $A' = g^s y_1^{-c}$ and $B' = h^s y_2^{-c}$. Finally, it outputs accept if and only if $c \stackrel{?}{=} H(PK, A', B', m)$.

It is not hard to see that the scheme is correct. Since $y_1 = g^x$ and $y_2 = h^x$, the signature verification algorithm computes $A' = g^s y_1^{-c} = g^{s-xc} = g^r = A$ and similarly for B'; thus, it is indeed the case that $H(PK, A', B', m) = c$.

THEOREM 1. *Let $\mathbb{G}$ be a (t', ε')-DDH group with $|\mathbb{G}| = q$ such that exponentiation in $\mathbb{G}$ takes time $t_{\exp}$. Then the above signature scheme is $(t, q_h, q_s, \varepsilon)$-secure (in the random oracle model), where:*

$$\begin{aligned} t &\leq t' - 2.4(q_s + 1)t_{\exp} \\ \varepsilon &\geq \varepsilon' + q_s q_h q^{-1} + (q_h + 1)q^{-1}. \end{aligned}$$

PROOF. Assume we have an algorithm $\mathcal{F}$, running in time at most t and making at most q_h hash queries and at most q_h signing queries, which forges a new message/signature pair with probability at least ε. We use $\mathcal{F}$ to construct an algorithm $\mathcal{D}$ running in time t' which solves the DDH problem with probability ε'. The stated result follows.

Algorithm $\mathcal{D}$ is given as input a tuple (g, h, y_1, y_2); its goal is to determine whether this represents a "random tuple" or a "DDH tuple". To this end, it sets $PK = (g, h, y_1, y_2)$ and runs $\mathcal{F}$ on input PK. Algorithm $\mathcal{D}$ simulates the signing oracle and the random oracle for $\mathcal{F}$ as follows:

Hash queries. In response to a query $H(PK, A, B, m)$, algorithm $\mathcal{D}$ first determines whether the output of H on this input has been previously set (either directly by a previous hash query, or as part of a signature query). If so, $\mathcal{D}$ returns the previously-assigned value. Otherwise, $\mathcal{D}$ responds with a value chosen uniformly at random from $\mathbb{Z}_q$.

Signing queries. If $\mathcal{F}$ asks for a signature on message m, then $\mathcal{D}$ attempts — in the standard way — to simulate a proof that (g, h, y_1, y_2) is a DDH tuple. Thus, $\mathcal{D}$ chooses random $c, s \in \mathbb{Z}_q$ and computes $A = g^s y_1^{-c}$ and $B = h^s y_2^{-c}$. If H had previously been queried on input (PK, A, B, m) then $\mathcal{D}$ aborts (with output 0); otherwise, $\mathcal{D}$ sets $H(PK, A, B, m) = c$ and outputs the signature (c, s)

At some point, $\mathcal{F}$ outputs its forgery $(\tilde{m}, \tilde{\sigma} = (\tilde{c}, \tilde{s}))$, where we assume that $\tilde{\sigma}$ was not previously the response to a query $\mathsf{Sign}_{SK}(\tilde{m})$. Letting $\tilde{A} = g^{\tilde{s}} y_1^{-\tilde{c}}$ and $\tilde{B} = h^{\tilde{s}} y_2^{-\tilde{c}}$, we also assume that $\mathcal{F}$ has previously queried $H(PK, \tilde{A}, \tilde{B}, \tilde{m})$. Now, if $H(PK, \tilde{A}, \tilde{B}, \tilde{m}) = c$ (i.e., $\mathsf{Vrfy}_{PK}(\tilde{m}, \tilde{\sigma}) = 1$) then $\mathcal{D}$ outputs 1; otherwise, $\mathcal{D}$ outputs 0.

We now analyze the probability that $\mathcal{D}$ outputs 1. If (g, h, y_1, y_2) is a Diffie-Hellman tuple, then $\mathcal{D}$ provides a perfect simulation for $\mathcal{F}$ except for the possibility of an abort. An abort can occur during $\mathcal{D}$'s simulation of any of the signing queries; in the simulation of any particular signing query, it is not hard to see that the probability of abort is at most $q_h/|\mathbb{G}|$. Thus the overall probability that $\mathcal{D}$ aborts is at most $q_s q_h/|\mathbb{G}|$. This means that $\mathcal{F}$ outputs a forgery (and hence $\mathcal{D}$ outputs 1) with probability at least $\varepsilon - q_s q_h/q$.

On the other hand, if (g, h, y_1, y_2) is a random tuple then with probability $1 - 1/q$ it is *not* a Diffie-Hellman tuple. In this case, for any query $H(PK, A, B, m)$ made by $\mathcal{F}$ there is at most one possible value of c for which there exists an s satisfying $A = g^s y_1^{-c}$ and $B = h^s y_2^{-c}$ (this is easy to see by looking at the two linear equations over $\mathbb{Z}_q$ in the exponent). Thus, $\mathcal{F}$ outputs a forgery (and hence $\mathcal{D}$ outputs 1) with probability at most $q^{-1} + q_h q^{-1}$.

Putting everything together, we see that

$$\begin{aligned} &|\Pr[x, y \leftarrow \mathbb{Z}_q : \mathcal{D}(g, g^x, g^y, g^{xy}) = 1] \\ &\qquad - \Pr[x, y, z \leftarrow \mathbb{Z}_q : \mathcal{D}(g, g^x, g^y, g^z) = 1]| \\ &\quad \geq \varepsilon - q_s q_h q^{-1} - q^{-1}(1 + q_h) \\ &\quad \geq \varepsilon'. \end{aligned}$$

The running time of $\mathcal{D}$ includes the running time of $\mathcal{F}$ and is otherwise dominated by the two multi-exponentiations that are performed for each query to the signing oracle plus those done in verifying the output of $\mathcal{F}$. Assuming (as in [18]) that a two-exponent multi-exponentiation takes time $1.2t_{\exp}$ gives the result of the theorem. □

REMARK. This technique of using the Fiat-Shamir heuristic to provide a *proof* rather than a *proof of knowledge* can be used more generally to achieve a tight security reduction based on a *decisional* problem rather than a non-tight security reduction based on a *computational* problem. As one example, the Fiat-Shamir signature scheme [16] includes values $\{y_i\}$ in the public key and the signer proves knowledge of the square roots of these values. Using the "forking lemma" of Pointcheval and Stern [24], we may thus obtain a non-tight security reduction for this scheme based on the hardness of computing square roots modulo N (which is equivalent to the hardness of factoring N). On the other hand, by having the signer prove that the $\{y_i\}$ are all quadratic residues (without necessarily proving knowledge of their square roots), we may obtain a *tight* security reduction based on the hardness of deciding quadratic residuosity.

3.2 Discussion

Our signature scheme is more efficient than the scheme analyzed by Goh and Jarecki when working over the same group $\mathbb{G}$. Signing in our scheme requires 2 exponentiations which may be pre-computed "off-line" before the message to

be signed is known; this may be compared to the 3 exponentiations required in the Goh-Jarecki scheme (one of which must be computed after the message is known). Verification in both schemes requires 2 multi-exponentiations. Our signature scheme vastly outperforms the Goh-Jarecki scheme in terms of signature length: signatures in our scheme are $2|q|$ bits, compared to $|p| + 2|q| + 111$ bits in [18] (in practice, $|p| \approx 1000$ while $|q| \approx 160$). Finally, an oft-neglected aspect of cryptographic protocols is their ease of implementation; here, too, we believe our scheme offers practical advantages. The Goh-Jarecki scheme requires two independent hash functions, one of which has $\mathbb{G}$ as its range; this seems more difficult to implement *correctly* than a single hash function mapping onto $\mathbb{Z}_q$. Finally, note that hashing onto $\mathbb{Z}_q$ is (in some cases) much faster than hasing onto $\mathbb{G}$. Indeed, hashing onto $\mathbb{Z}_q$ requires only a (small) constant number of hash evaluations. On the other hand, when $\mathbb{G}$ is an order-q subgroup of $\mathbb{Z}_p^*$ (with $p = \alpha q + 1$), hashing onto $\mathbb{G}$ requires computing an exponentiation to the power α modulo p. Assuming $|p| = 1024, |q| = 160$, and thus $|\alpha| \approx 864$ (which are parameters often used in practice), computing the required hash (which uses an 864-bit exponent) is even more expensive than an exponentiation in $\mathbb{G}$ itself (which uses only 160-bit exponents)!

Interestingly, we also obtain a tighter security reduction than Goh and Jarecki: assuming t_{exp} is 100 times larger than the time to evaluate H and letting $t = q_h = 2^{60}$, $q_s = 2^{30}$, and $|q| \geq 160$, we obtain the (essentially) optimal:

$$t' \approx t, \;\; \varepsilon' \approx \varepsilon,$$

whereas Goh-Jarecki [18] obtain a (small) 2^8 decrease in security. Thus, our reduction carries *no decrease in security*, and our signature scheme is as hard to break as the underlying DDH problem is to solve. Of course, the security of our scheme is based on the (potentially) easier DDH problem rather than the CDH problem. Still, *given current techniques* the DDH problem is as hard as the CDH problem (and both are as hard as the discrete logarithm problem) for a number of widely-used groups [4].

4. AVOIDING THE RANDOM SALT IN PSS

We begin with an informal recap of previous work on FDH, PSS, and probabilistic FDH (PFDH); PFDH was introduced by Coron [12] as a "simplified" version of PSS which highlights the key features of PSS while being slightly less efficient in general (yet, the "PFDH methodology" was adopted by Goh and Jarecki to improve the security of their scheme [18]). Throughout, we let H denote a random oracle mapping strings to the appropriate range.

FDH was originally defined [1] for an arbitrary trapdoor permutation. Key generation simply generates a trapdoor permutation f as the public key, and the secret key is the associated trapdoor allowing efficient computation of f^{-1}. To sign message m, the signer outputs $\sigma = f^{-1}(H(m))$; verification is done in the obvious way. Bellare and Rogaway showed that if f cannot be inverted with probability better than ε', then forgery with probability better than $\varepsilon \approx (q_h + q_s)\varepsilon'$ is impossible. In their proof, the simulator "guesses" the index i of the hash query which results in a forgery; the simulator then sets the output of the i^{th} hash query $H(m_i)$ to be y (where y is the value to be inverted). The output of all other hash queries $H(m_j)$ is set to $f(x_j)$ for random x_j. In this way, the simulator can answer all signing queries except those for message m_i; furthermore, if the forger outputs a signature on m_i, the simulator obtains the desired inverse. However, having to guess the index i (from among all $(q_s + q_h)$ queries to H) results in a substantial loss of security.

Subsequently, Coron [11] improved the exact security of FDH for the specific case when RSA is the underlying trapdoor permutation by using random self-reducibility properties of RSA (but see footnote 1). Here, to invert a given value y the simulator answers hash query $H(m_i)$, for all i, with $f(x_i) = x_i^e$ (with probability ρ) and with $x_i^e y$ (with probability $1 - \rho$); note that the simulator can answer signing queries related to hash queries of the first type, while forgeries related to hash queries of the second type allow computation of the desired inverse. By appropriately choosing ρ, the tighter security bound $\varepsilon \approx q_s\varepsilon'$ can be obtained. Interestingly, this proof technique was generalized by Dodis and Reyzin [13] who showed that whenever the trapdoor permutation f is induced by a claw-free permutation (i.e., f arises from a pair of claw-free permutations (f, g)), then FDH achieves security $\varepsilon \approx q_s\varepsilon'$. In their proof, the simulator answers hash queries with $f(x_i)$ (with probability ρ) and with $g(x_i)$ (with probability $1 - \rho$); as before, the simulator can answer signing queries related to hash queries of the first type, while forgeries related to hash queries of the second type yield a "claw", as desired.

In FDH, each message has a unique signature. Thus, in the security proofs for FDH, for each message m the simulator either can produce a signature of m (in which case a forged signature on m does not "help") or cannot produce a signature of m (in which case a forged signature on m breaks the underlying assumption, but a signing request for m requires the simulator to abort). This motivated the design of PSS [3] and PFDH[3] [12] in which each message has multiple valid signatures. We discuss PFDH, but the ideas apply to PSS as well. In PFDH, key generation is done as before; to sign m, the signer chooses random "salt" r and outputs $\langle r, f^{-1}(H(r\|m))\rangle$. Verification is done in the obvious way. If f is induced by claw-free permutation (f, g) then, in the proof of security [12, 13], the simulator can answer *all* hash queries with $g(x_i)$ and thus any forgery (giving a value $f^{-1}(g(x_i))$) yields a "claw"; still, the simulator can sign any message m as long as it picks a random salt r such that $H(r\|m)$ was not previously queried (because it can then set $H(r\|m) = f(x_i)$ and output the signature $\langle r, x_i\rangle$).

This gives a perfect simulation unless, in answering a signing query for message m, the simulator chooses r such that $H(r\|m)$ *was* previously queried. But setting $|r|$ long enough ensures that this occurs with small probability. Bellare and Rogaway [3] showed that having $|r| \geq 2\log_2 q_h + \log_2 \varepsilon'$ is sufficient to obtain a tight security proof; Coron [12] improved this to $|r| \geq \log_2 q_s$. Note that reducing the length of r reduces the amount of randomness required and also results in a shorter signature.

4.1 Tight Security with No Random Salt

All previous work requires *some* random salt in order to achieve a tight security proof. Here, we show a very simple modification with a tight security reduction which avoids the random salt *altogether*. In order to avoid the difficulties that arise in the proof of security for FDH (and to circumvent the

[3]As noted above, PFDH simplifies PSS yet distills the interesting features of the security proof.

fact [12, Theorem 5] that tight reductions are impossible for FDH-like schemes in which each message has a unique signature), our scheme has the property that each message has *two* possible signatures; in particular, our scheme may be viewed as PFDH with $|r| = 1$. However, in contrast to PFDH where a random valid signature is chosen by the signer each time a message is signed, in our scheme only one valid signature (for a given message) is ever produced by the signer. Somewhat surprisingly, this is enough to obtain a tight security reduction!

We now more carefully describe our scheme and give a full proof of security. As in [13], our schemes assume a trapdoor permutation f induced by a claw-free permutation (f, g). In the description, H is a random oracle mapping arbitrary-length strings to the appropriate range.

$\underline{\mathsf{Gen}}$ runs $\mathsf{cf\text{-}Gen}$ to obtain (f, g) and trapdoor information td. The public key is $PK = f$ and the secret key is td.

$\underline{\mathsf{Sign}_{SK}(m)}$ (where $m \in \{0,1\}^*$) first checks to see whether m has been signed before (below, we show a simple way to avoid maintaining state); if it has, the previously-generated signature is output. Otherwise, the signer chooses a random bit b and outputs $\sigma = f^{-1}(H(b\|m))$.

$\underline{\mathsf{Vrfy}_{PK}(m, \sigma)}$ accepts if and only if either $f(\sigma) \stackrel{?}{=} H(0\|m)$ *or* $f(\sigma) \stackrel{?}{=} H(1\|m)$.

Efficiency improvements. Before giving the proof of security, we note some immediate enhancements of the scheme above. First of all, to avoid having the verifier compute both $H(0\|m)$ and $H(1\|m)$, the signer can include the bit b along with the signature. Second, to avoid having the signer maintain state (i.e., a record of all previous message/signature pairs) we can simply have the signer generate b as a pseudorandom function of m; since we are working in the random oracle model anyway, the simplest solution is to set $b = H'(SK\|m)$ where $H' : \{0,1\}^* \to \{0,1\}$ is an independent random oracle. It is also possible to have the signer store a "short" random seed $s \in \{0,1\}^{80}$ which is used to derive b via $H'(s\|m)$. (As mentioned earlier, using a PRF and setting $b = F_s(m)$ would also suffice.) Finally, in case the original message $\tilde{m}$ is very long, we can avoid computing two hashes over long inputs by using a collision-resistant hash function H'' to compute $m = H''(\tilde{m})$ and then using the resulting m in the above construction.

Implementation using RSA. We noted earlier that the RSA trapdoor permutation may be viewed as being induced by a claw-free permutation. In this case, the above scheme is almost as simple as FDH: the user's public key is N, e and the secret key contains d such that $ed = 1 \bmod \varphi(N)$. To sign message m, the signer sets $b = H'(SK\|m)$ and computes $\sigma = (H(b\|m))^d \bmod N$. Verification requires a single exponentiation and (at most) two hash function evaluations.

THEOREM 2. *If the claw-free permutation in the scheme above is (t', ε')-secure and the time to compute f or g is at most t_{f}, then the above signature scheme is $(t, q_h, q_s, \varepsilon)$-secure (in the random oracle model), where:*

$$\begin{aligned} t &\leq t' - (q_h + q_s) \cdot t_{\mathrm{f}} \\ \varepsilon &\geq 2\varepsilon'. \end{aligned}$$

PROOF. Assume we have an algorithm $\mathcal{F}$, running in time at most t and making at most q_h hash queries and at most q_s signing queries, which forges a new message/signature pair with probability at least ε. We use $\mathcal{F}$ to construct an algorithm $\mathcal{I}$ running in time at most t' which finds a claw with probability at least ε'. The stated result follows.

Algorithm $\mathcal{I}$ is given the pair (f, g) and its goal is to output a claw (i.e., x_0, x_1 such that $f(x_0) = g(x_1)$). It sets $PK = f$ and runs $\mathcal{F}$ on input PK. Algorithm $\mathcal{I}$ simulates the signing oracle and the random oracle for $\mathcal{F}$ as follows:

Global state. Whenever a hash query $H(b\|m)$ or a signing query $\mathsf{Sign}_{SK}(m)$ is made for a *new* message m, algorithm $\mathcal{I}$ chooses a random $b_m \in \{0,1\}$. Informally, this represents the fact that $\mathcal{I}$ will be able to produce the value $f^{-1}(H(b_m\|m))$. We do not include this step explicitly in the descriptions that follow.

Hash queries. $\mathcal{I}$ will maintain a list HL of tuples whose exact format we will describe below. In response to a query $H(b\|m)$, algorithm $\mathcal{I}$ proceeds as follows: if there is a tuple (b, m, x, y) in HL, then $\mathcal{I}$ returns y. Otherwise, $\mathcal{I}$ first chooses random x from the appropriate domain. If $b = b_m$, then $\mathcal{I}$ returns $y = f(x)$; if $b \neq b_m$ then $\mathcal{I}$ returns $y = g(x)$. In either case, $\mathcal{I}$ then stores (b, m, x, y) in HL.

Signing queries. If $\mathcal{F}$ asks for a signature on a message m, algorithm $\mathcal{I}$ computes $y = H(b_m\|m)$ and finds the tuple (b_m, m, x, y) in HL. It then returns $\sigma = x$. (Note that $f(\sigma) = H(b_m\|m)$, so this is indeed a valid signature.)

$\mathcal{I}$ provides $\mathcal{F}$ with a perfect simulation; in particular, all signing queries are answered with valid signatures. At some point, $\mathcal{F}$ outputs its forgery $(\widetilde{m}, \widetilde{\sigma})$, where we assume that both $H(0\|\widetilde{m})$ and $H(1\|\widetilde{m})$ were previously queried and that $\widetilde{\sigma}$ was not the previous response to a query $\mathsf{Sign}_{SK}(\widetilde{m})$. If $\mathsf{Vrfy}_{PK}(\widetilde{m}, \widetilde{\sigma}) = 1$, there are two possible cases:

Case 1: $f(\widetilde{\sigma}) = H(b_{\widetilde{m}}\|\widetilde{m})$. In this case, $\mathcal{I}$ cannot find a claw and simply aborts.

Case 2: $f(\widetilde{\sigma}) = H(1 - b_{\widetilde{m}}\|\widetilde{m})$. In this case, $\mathcal{I}$ finds a claw as follows: it finds a tuple $(1 - b_{\widetilde{m}}, \widetilde{m}, x, y)$ in HL and outputs the claw $(\widetilde{\sigma}, x)$. (Recall that $g(x) = y = H(1 - b_{\widetilde{m}}\|\widetilde{m})$.)

Since the value of $b_{\widetilde{m}}$ is information-theoretically hidden from $\mathcal{F}$, the forgery will enable $\mathcal{I}$ to find a claw with probability $1/2$. Thus, if $\mathcal{F}$ outputs a forgery with probability at least ε then $\mathcal{I}$ outputs a claw with probability at least $\varepsilon/2$. The running time of $\mathcal{I}$ includes the running time of $\mathcal{F}$ and is otherwise dominated by the computation of f or g each time a hash query is answered. □

REMARK. In addition to achieving a tight security proof, our scheme has the advantage of being *deterministic*. Having a deterministic signing algorithm is advantageous in a number of environments.

4.2 A Scheme Supporting Message Recovery

In the last section, we showed an FDH-/PFDH-like scheme achieving a tight security reduction without using a random salt. We now apply the same idea to PSS-R to obtain a signature scheme (with tight security proof) which supports message recovery. By avoiding the use of a random salt, our technique enables recovery of longer messages than previous schemes. That is (for the case of 1024-bit RSA), whereas

the original analysis of PSS-R [3] allowed recovery of 663-bit messages and Coron's analysis extended this to recovery of 813-bit messages, the scheme below can be used to sign 843-bit messages at the same level of security.

We describe our scheme using RSA; however, the construction and proof may be generalized for arbitrary claw-free permutations. For the sake of generality, we parameterize our scheme by k and k_1 where k is the length (in bits) of the modulus used and $k - k_1 - 1$ is the length of the messages to be signed; we achieve the result stated in the previous paragraph by setting $k = 1024$ and $k_1 = 180$. In the description below, $H : \{0,1\}^* \to \{0,1\}^{k_1}$ and $G : \{0,1\}^{k_1} \to \{0,1\}^{k-k_1-1}$ are hash functions which will be modeled as independent random oracles.

$\underline{\mathsf{Gen}}$ generates a k-bit modulus N along with public exponent e and private exponent d such that $ed = 1 \bmod \varphi(N)$. The public key is $PK = (N, e)$ and secret key is d.

$\underline{\mathsf{Sign}_{SK}(m)}$ (where $m \in \{0,1\}^{k-k_1-1}$) first checks to see whether m has been signed before; if it has, the previously-generated signature is output. (In the previous section, we describe a simple way to avoid the need for the signer to maintain state.) Otherwise, the signer chooses a random $b \in \{0,1\}$, computes $w = H(b\|m)$ and $r^* = G(w) \oplus m$, and sets $\widetilde{y} = 0\|w\|r^*$. Finally, the signer outputs signature $\sigma = \widetilde{y}^d \bmod N$.

$\underline{\mathsf{Vrfy}_{PK}(\sigma)}$ parses PK as (N, e). It computes $\widetilde{y} = \sigma^e \bmod N$ and parses $\widetilde{y}$ as $0\|w\|r^*$. It then sets $m = G(w) \oplus r^*$ and accepts the message m if and only if either $H(0\|m) \stackrel{?}{=} w$ or $H(1\|m) \stackrel{?}{=} w$ (as above, we can include the appropriate value of b with the signature if desired).

THEOREM 3. *If RSA with k-bit moduli is a (t', ε')-secure trapdoor permutation and computing e^{th} powers modulo a k-bit modulus can be done in time t_{exp}, then the above signature scheme is $(t, q_h, q_s, \varepsilon)$-secure (in the random oracle model), where:*

$$\begin{aligned} t &\leq t' - (q_h + q_s) \cdot (k_1 - \log_2 q_h) \cdot t_{\text{exp}} \\ \varepsilon &\geq 2\varepsilon' + 2 \cdot (q_h + q_s)^2 \cdot 2^{-k_1}. \end{aligned}$$

The proof is similar to the proof of Theorem 2 (cf. also the proofs of PSS-R in the full version of [12] and of Theorem 4, below), and is therefore omitted in the present abstract.

The theorem shows that to achieve tight security, we require $(q_h + q_s)^2 2^{-k_1} \approx \varepsilon'$ or, equivalently, $k_1 \approx 2 \cdot \log_2(q_h + q_s) + \log_2 \frac{1}{\varepsilon'}$. Taking $q_h \approx 2^{60}$, $q_s \approx 2^{30}$, and $\varepsilon' \approx 2^{-60}$ (as suggested by [12]) shows that the length of k_1 must be roughly 180 bits.

4.3 Message Recovery With Optimal Message Length

We may apply a technique suggested by Granboulan [20] to obtain a variant of PSS-R which is essentially *optimal* in terms of the allowable message length (an easy proof of optimality is given below). Unfortunately, this construction requires the *random permutation model*[4] which seems stronger than the random oracle model; also, we are not aware of any appropriate way to instantiate the random permutation for large block sizes (i.e., block sizes larger than the block size of a cipher such as AES).

Here, we describe our construction using a generic claw-free permutation and assume further that the domain of the permutation is $\{0,1\}^k$; however, the scheme can be easily adapted for the case of, e.g., RSA as in the previous section (see also [3, 12, 20]). We let $k - k_1$ denote the length of the messages to be signed, and let $E : \{0,1\}^k \to \{0,1\}^k$ denote a public, random permutation with (public) inverse E^{-1}.

$\underline{\mathsf{Gen}}$ runs $\mathsf{cf\text{-}Gen}$ to obtain (f, g) and trapdoor information td. The public key is $PK = f$ and the secret key is td.

$\underline{\mathsf{Sign}_{SK}(m)}$ (where $m \in \{0,1\}^{k-k_1}$) first checks to see whether m has been signed before; if it has, the previously-generated signature is output. (As in the previous two sections, it is easy to modify the scheme to avoid maintaining state.) Otherwise, the signer chooses a random bit b and outputs $\sigma = f^{-1}(E(b^{k_1}\|m))$.

$\underline{\mathsf{Vrfy}_{PK}(\sigma)}$ computes $v\|m = E^{-1}(f(\sigma))$ where $v \in \{0,1\}^{k_1}$; it outputs m if $v \in \{0^{k_1}, 1^{k_1}\}$ and otherwise outputs reject.

THEOREM 4. *If the claw-free permutation in the scheme above is (t', ε')-secure and the time to compute f or g is at most t_{f}, then the above signature scheme is $(t, q_E, q_s, \varepsilon)$-secure (in the random permutation model), where:*

$$\begin{aligned} t &\leq t' - (q_E + q_s) \cdot t_{\text{f}} \\ \varepsilon &\geq 2\varepsilon' + 2 \cdot q_E \cdot 2^{-k_1}. \end{aligned}$$

(Here, q_E is the number of queries made by the adversary to the E/E^{-1} oracles.)

PROOF. Assume we have an algorithm $\mathcal{F}$, running in time at most t and making at most q_E queries to E/E^{-1} and at most q_s signing queries, which forges a new signature with probability at least ε. We use $\mathcal{F}$ to construct an algorithm $\mathcal{I}$ running in time at most t' which finds a claw with probability at least ε. The states result follows.

Algorithm $\mathcal{I}$ is given as input (f, g) and its goal is to output a claw (i.e., values x_0, x_1 such that $f(x_0) = g(x_1)$). It sets $PK = f$ and runs $\mathcal{F}$ on input PK. Algorithm $\mathcal{I}$ simulates the signing oracle and the oracles E/E^{-1} for $\mathcal{F}$ as follows:

Global state. Whenever a query $E(v\|m)$ or $\mathsf{Sign}_{SK}(m)$ is made for a *new* message m, algorithm $\mathcal{I}$ chooses a random $b_m \in \{0,1\}$. Informally, this represents the fact that $\mathcal{I}$ will be able to produce the value $f^{-1}(E(b_m^{k_1}\|m))$. We do not include this step explicitly in the descriptions that follow.

Queries to E/E^{-1}. Algorithm $\mathcal{I}$ will maintain the lists $PAIR$, L, and R with the following purpose: $PAIR$ will consist of tuples (w, w', x) such that $E(w) = w'$ (equivalently, $E^{-1}(w') = w$). List L will consist of all points w such that $E(w)$ is defined, and R will consist of all points w' such that $E^{-1}(w')$ is defined.

To answer the query $E^{-1}(w')$, algorithm $\mathcal{I}$ checks whether $w' \in R$. If so, it then finds the unique tuple $(w, w', x) \in PAIR$ and outputs w. Otherwise, it picks a random value $w \in \{0,1\}^k \setminus L$ (recall that $\{0,1\}^k$ is assumed to be the domain of f_0 and f_1). It parses w as $v\|m$ with $v \in \{0,1\}^{k_1}$; if $v \in \{0^{k_1}, 1^{k_1}\}$, then $\mathcal{I}$ aborts. If not, $\mathcal{I}$ returns w to $\mathcal{F}$, adds w to L, adds w' to R, and adds $(w, w', \perp)$ to $PAIR$.

[4] The random permutation model assumes a public, random permutation $E(\cdot)$ (along with its inverse $E^{-1}(\cdot)$) to which all parties have oracle access. This model is weaker than the often-used *ideal cipher* model, which assumes a public, keyed cipher $E(\cdot,\cdot)$ (and its inverse) such that $E(k,\cdot)$ is an independent, random permutation for each key k.

To answer the query $E(w)$, algorithm $\mathcal{I}$ checks whether $w \in L$. If so, it then finds the unique tuple $(w, w', x) \in PAIR$ and outputs w'. Otherwise, it parses w as $v\|m$ with $v \in \{0,1\}^{k_1}$ and chooses random $x \in \{0,1\}^k$. If $v = b_m^{k_1}$, it sets $w' = f(x)$; otherwise, it sets $w' = g(x)$. If $w' \in R$, then $\mathcal{I}$ aborts. Otherwise, it returns the result w' to $\mathcal{F}$, adds w to L, adds w' to R, and adds (w, w', x) to $PAIR$.

Signing queries. When $\mathcal{F}$ asks for a signature on a message m, algorithm $\mathcal{I}$ first checks whether a signature on m has been requested previously; if so, $\mathcal{I}$ simply outputs the same signature that was output before. Otherwise, $\mathcal{I}$ first computes $y = E(b_m^{k_1}\|m)$. It then finds the tuple of the form $(b_m^{k_1}\|m, y, x)$ in $PAIR$ and answers the query with x. It is crucial to note that as long as $\mathcal{I}$ has not aborted by this point, we have $x \neq \perp$.

As long as $\mathcal{I}$ does not abort, it provides $\mathcal{F}$ with a perfect simulation (in particular, all signing queries of $\mathcal{F}$ can be answered with valid signatures). If $\mathcal{I}$ never aborts, then at some point $\mathcal{F}$ outputs its forgery $\widetilde{\sigma}$; we assume that $f(\widetilde{\sigma}) \in R$ and that $\widetilde{\sigma}$ was not the response to a previous signing query. If $\widetilde{\sigma}$ is a valid signature, it means that there is a unique tuple of the form $(b^{k_1}\|\widetilde{m}, f(\widetilde{\sigma}), x)$ in $PAIR$; the crux of the proof is to notice that since $\mathcal{I}$ has not aborted, we have $x \neq \perp$. With probability $1/2$ it is the case that $b \neq b_{\widetilde{m}}$ in which case $\mathcal{I}$ can output the claw $(\widetilde{\sigma}, x)$.

To complete the proof, we need only analyze the probability that $\mathcal{I}$ aborts. An abort can occur in one of two ways: first, $\mathcal{F}$ might make a query $E^{-1}(w')$ which results in an output having k_1 leading "0" or "1" bits. Second, when answering a query to the E-oracle, a collision may occur with a previously defined value; this ruins the simulation since E is supposed to be a permutation. An abort of the first type occurs with probability bounded from above by $2 \cdot q_E \cdot \frac{2^{k-k_1}}{2^k - q_E}$, which, for values of k and q_E encountered in practice (i.e., $k \approx 1024$ and $q_E \approx 2^{60}$) is well-approximated by $2 \cdot q_E \cdot 2^{-k_1}$. An abort of the second type occurs with probability bounded from above by $(q_E + q_s)^2 \cdot 2^{-k}$. For values of q_E, q_s, and k encountered in practice, this term is entirely negligible.

Putting everything together, we see that if $\mathcal{F}$ forges signatures with probability ε, then $\mathcal{I}$ obtains a claw with probability (roughly) $\frac{1}{2} \cdot (\varepsilon - 2 \cdot q_E \cdot 2^{-k_1})$. Furthermore, the running time of $\mathcal{I}$ includes the running time of $\mathcal{F}$ and is otherwise dominated by the computation of f or g each time a permutation query is answered. □

To achieve tight security, we require $q_E \cdot 2^{-k_1} \approx \varepsilon'$ or, equivalently, $k_1 \approx \log_2(q_E/\varepsilon')$. Taking $q_E \approx 2^{60}$ and $\varepsilon' \approx 2^{-60}$ (following [12]) we see that k_1 must be 120 bits long. In other words, the signature scheme requires only 120 bits of redundancy and can sign messages of length $k - 120$. It is not hard to see that this result is essentially optimal in terms of the amount of redundancy used, and we formalize this in the following lemma.

LEMMA 1. *Let* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *be a signature scheme supporting message recovery for messages of length* ℓ *in which signatures are of length* s*. Then there exists an adversary running in time* $O(t)$ *which forges a signature with probability roughly* $t \cdot 2^{\ell - s}$*.*

Before giving the (simple) proof, it is worth noting that we do not restrict the forger in any way, do not assume any "black-box" access to the forger, and do not make any mention of random oracles or ideal permutations. From a practical point of view, the lemma indicates that if we want to achieve probability of forgery 2^{-60} against adversaries running in time 2^{60}, then 120 bits of redundancy (i.e., $s - \ell$) are *necessary*.

PROOF. Consider the following forger $\mathcal{F}$: Given a public key PK, adversary $\mathcal{F}$ picks $\sigma \in \{0,1\}^s$ and checks whether $\mathsf{Vrfy}_{PK}(\sigma) \neq \mathsf{reject}$. Since every message in $\{0,1\}^\ell$ corresponds to at least one signature, and since there can be no signature which corresponds to two different messages (since the scheme supports message recovery), there are at least 2^ℓ valid signatures. Thus, picking σ at random yields a valid signature with probability at least $2^{\ell - s}$. Repeating this process at most t times allows $\mathcal{F}$ to achieve the stated bound. □

4.4 Further Applications

We show how the technique of the preceding sections can be applied to eliminate the need for a random salt in schemes based on the CDH assumption (even though the CDH problem does not immediately give rise to a claw-free permutation). We focus on the schemes of [7, 18] which both have a similar, high-level structure: the public key contains elements g, h in some group $\mathbb{G}$, and the secret key contains the value $x = \log_g h$. Signing a message m involves computing $\sigma = (H(m))^x$, where H is a random oracle mapping strings uniformly onto $\mathbb{G}$.

Boneh, Lynn, and Shacham [7] use an "FDH-like" approach to analyze their scheme (but with the optimization of Coron [11]); consequently they lose a factor of q_s in the security reduction. On the other hand, Goh and Jarecki [18] use a "PFDH-like" approach and include the random salt r with the resulting signature; this enables them to obtain a tight security reduction at the expense of increasing the signature length.

We show here how to achieve a tight security reduction without any random salt. The public key and the secret key are as before but signing message m now involves choosing random b and computing $\sigma = (H(b\|m))^x$ (and using the same b to answer any subsequent signing queries for the same message). Without going through the details of the proof (which parallel those of the proof of Theorem 2), note that a simulator who wants to solve an instance of the CDH problem (g, h, y) simply sets the public key to be $PK = (g, h)$. Whenever there is a query on a new message m, it chooses a random bit $b_m \in \{0,1\}$. It will then answer the hash query $H(b\|m)$ with g^α if $b = b_m$, or with yg^α if $b \neq b_m$. Now, the simulator can correctly answer all signing requests by computing $y = H(b_m\|m)$, looking up the value of $\alpha = \log_g y$, and setting $\sigma = h^\alpha$. Conversely, any valid forgery results in a solution to the original CDH instance with probability $1/2$: if the forgery includes $\widetilde{\sigma} = (H(b\|m))^x$ for $b \neq b_m$, then the simulator knows α such that $H(b\|m) = yg^\alpha$; therefore, $\widetilde{\sigma} = y^x h^\alpha$ from which the simulator can compute the desired answer y^x.

REMARK. The above techniques extend to allow a tighter proof of security for (a variant of) the *identity-based encryption scheme* of Boneh and Franklin [6]. We do not discuss the details, but instead assume the reader is familiar with the Boneh-Franklin construction and give only an overview of the approach. In the original Boneh-Franklin

scheme, for any identity ID there is an associated "public key" $PK_{ID} = H(ID)$; anyone can encrypt messages to user ID using some master parameters and PK_{ID} (which can be derived simply using ID, making the scheme "identity-based"). There is also a corresponding secret key SK_{ID} that the private-key generator (PKG) gives to user ID to enable decryption. In the model of security, an adversary is allowed to request ("expose") secret keys for identities $\mathcal{I} = \{ID_1, \ldots, ID_\ell\}$ of his choice, yet encryption to any user $ID \notin \mathcal{I}$ should remain secure.

In the given proof of security [6], a simulator needs to "guess" (in some sense) which identities the adversary will expose. This leads to a loss of $O(q_e)$ in the security (where q_e is the number of key exposure queries).

Consider now a modified scheme in which for any ID there are *two* "public keys" $PK_{ID,0} = H(0\|ID)$ and $PK_{ID,1} = H(1\|ID)$; furthermore, to encrypt a message to user ID, a sender now encrypts the same message with respect to *both* of these public keys. The PKG, however, only gives to ID *one* of the corresponding private keys (i.e., either $SK_{ID,0}$ or $SK_{ID,1}$ but not both). Note that a single such key is sufficient to enable correct decryption. Following the proof techniques discussed in this paper, a simulation can be set up in which the simulator knows *exactly one* secret key for *every* ID. This allows *all* key exposure queries to be answered by the simulator, while ensuring that encryption to any non-exposed ID remains secret. Thus, this modification enables a tight proof of security at the cost of reducing the efficiency of encryption by a factor of two.

Acknowledgments

The remark at the end of Section 3.1 is due to Yevgeniy Dodis, while the remark at the end of Section 4.4 is due to Dan Boneh. We also thank Dan for suggesting to hash long messages only a single time in the course of signing (cf. Section 4.1). We also appreciate the remarks of Stanislaw Jarecki, who pointed out that hashing onto $\mathbb{Z}_q$ is often much more efficient than hashing onto $\mathbb{G}$ (cf. Section 3.2). Finally, we thank the anonymous referees for their helpful comments.

5. REFERENCES

[1] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. *ACM Conference on Computer and Communications Security*, 1993.

[2] M. Bellare and P. Rogaway. Optimal asymmetric encryption padding. Eurocrypt '94.

[3] M. Bellare and P. Rogaway. The exact security of digital signatures — how to sign with RSA and Rabin. Eurocrypt '96.

[4] D. Boneh. The decision Diffie-Hellman problem. ANTS '98.

[5] D. Boneh. Simplified OAEP for the RSA and Rabin functions. Crypto 2001.

[6] D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. Crypto 2001. Full version to appear in *SIAM J. Computing* and available at http://eprint.iacr.org/2001/090.

[7] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. Asiacrypt 2001.

[8] E. Brickell, D. Pointcheval, S. Vaudenay, and M. Yung. Design validations for discrete logarithm based signature schemes. PKC 2000.

[9] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. STOC '98.

[10] D. Chaum and H. van Antwerpen. Undeniable signatures. Crypto '89.

[11] J.-S. Coron. On the exact security of full-domain hash. Crypto 2000.

[12] J.-S. Coron. Optimal security proofs for PSS and other signature schemes. Eurocrypt 2002. Full version available at http://eprint.iacr.org/2001/062/.

[13] Y. Dodis and L. Reyzin. On the power of claw-free permutations. Security in Communication Networks 2002.

[14] T. El Gamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Info. Theory* 31(4): 469–472 (1985).

[15] *Federal Information Processing Standards* publication #186-2. 2000. Digital signature standard (DSS). U.S. Department of Commerce/National Institute of Standards and Technology.

[16] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. Crypto '86.

[17] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. Crypto 2001.

[18] E.-J. Goh and S. Jarecki. A signature scheme as secure as the Diffie-Hellman problem. Eurocrypt 2003.

[19] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing* 17(2): 281–308 (1988).

[20] L. Granboulan. Short signatures in the random oracle model. Asiacrypt 2002.

[21] J. Jonsson. An OAEP variant with a tight security proof. Available at http://eprint.iacr.org/2002/034/.

[22] U. Maurer and S. Wolf. The Diffie-Hellman protocol. *Designs, Codes, and Cryptography* 19(2/3): 147–171 (2000).

[23] S. Micali and L. Reyzin. Improving the exact security of digital signature schemes. *J. Cryptology* 15(1): 1–18 (2002).

[24] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology* 13(3): 361–396 (2000).

[25] C. Schnorr. Efficient identification and signatures for smart cards. Crypto '89.

[26] V. Shoup. Lower bounds for discrete logarithms and related problems. Eurocrypt '97.

[27] V. Shoup. OAEP reconsidered. Crypto 2001.

[28] V. Shoup. A proposal for an ISO standard for public-key encryption. Available at http://eprint.iacr.org/2001/112.