On the Security of RSA-PSS in the Wild

Saqib A. Kakvi

October 31, 2019

Department of Computer Science, Bergische Universität Wuppertal
kakvi@uni-wuppertal.de

Abstract

The RSA Probabilistic Signature Scheme (RSA-PSS) due to Bellare and Rogaway (EUROCRYPT 1996) is a widely deployed signature scheme. In particular it is a suggested replacement for the deterministic RSA Full Domain Hash (RSA-FDH) by Bellare and Rogaway (ACM CCS 1993) and PKCS#1 v1.5 (RFC 2313), as it can provide stronger security guarantees. It has since been shown by Kakvi and Kiltz (EUROCRYPT 2012, Journal of Cryptology 2018) that RSA-FDH provides similar security to that of RSA-PSS, also in the case when RSA-PSS is not randomized. Recently, Jager, Kakvi and May (ACM CCS 2018) showed that PKCS#1 v1.5 provides comparable security to both RSA-FDH and RSA-PSS. However, all these proofs consider each signature scheme in isolation, where in practice this is not the case. The most interesting case is that in TLS 1.3, PKCS#1 v1.5 signatures are still included for reasons of backwards compatibility, meaning both RSA-PSS and PKCS#1 v1.5 signatures are implemented. To save space, the key material is shared between the two schemes, which means the aforementioned security proofs no longer apply. We investigate the security of this joint usage of key material in the context of Sibling Signatures, which were introduced by Camenisch, Drijvers, and Dubovitskaya (ACM CCS 2017). It must be noted that we consider the standardised version of RSA-PSS (IEEE Standard P1363-2000), which deviates from the original scheme considered in all previous papers. We are able to show that this joint usage is indeed secure, and achieves a security level that closely matches that of PKCS#1 v1.5 signatures and that both schemes can be safely used, if the output lengths of the hash functions are chosen appropriately.

 The definitive Version of Record was published in 2019 Security Standardisation Research Conference (SSR'19), http://dx.doi.org/10.1145/3338500.3360333.

1 Introduction

The RSA PKCS#1 v1.5 signature algorithm dates back to 1991, when it was published at the NIST/OSI Implementors' Workshop (SEC-SIG-91-18). It was then first publically specified in 1998 in [RFC 2313]. The scheme was, however, not accompanied by a security proof, which later raised some concerns: the accompanying PKCS#1 v1.5 encryption scheme from [RFC 2313] was shown to be completely insecure, mainly based on the Bleichenbacher padding oracle attacks [Ble98]. While these attacks, which include but are not limited to [CFPR96, CJNP00, KPR03, DLP+12, BFK+12, ZJRR14, JSS15a, JSS15b, MSW+14, BSY17], did not directly break the PKCS#1 v1.5 signature schemes, they did expose some potential weaknesses. As a response to this, the standard was expanded to include the RSA-Probabilistic Signature Scheme (RSA-PSS) due to Bellare and Rogaway [BR96]. A variant of the original PSS scheme [IEEE 1363], was brought in as a side-by-side alternative in version 2.1 [RFC 2437], but was later upgraded to the preferred algorithm in version 2.2:

> "Although no attacks are known against RSASSA-PKCS1-v1_5, in the interest of increased robustness, RSASSA-PSS is REQUIRED in new applications. RSASSA-PKCS1-v1_5 is included only for compatibility with existing applications." [RFC 8017, Sec. 8]

The standardised variant deviated sufficiently from the original proposal so that the proof no longer holds. A discussion of the changes made and the rationale behind them, along with a proof of security was presented by Jonsson [Jon01]. We briefly recap the changes made to the scheme, and the effect on the security proof in Section 3.1.

While it was generally agreed that RSA-PSS should be adopted as the only RSA-based signature algorithm, this was easier said than done. Most notably, when the topic was raised in the context of TLS 1.3 [RFC 8446], there was some discussion regarding this[1]. In the end, PKCS#1 v1.5 signatures were kept in TLS 1.3 for backwards compatibility, as it is still widely deployed but RSA-PSS was to be the preferred signature scheme. This is due to the fact that RSA-PSS comes with a *(tight) proof of security*, more or less out of the box.

This meant that both algorithms had to be implemented side-by-side and while this might sound like piling a secure signature scheme unto an insecure one, this is not the case. Recently, Jager et al. [JKM18a, JKM18b] presented a security proof for some variants of the PKCS#1 v1.5 signatures, providing some confidence in their use. At this point, one would hope that simply implementing both schemes in line with the known security proofs would guarantee security, but that is not the case. The proofs assume that the schemes are used in isolation, with no additional usage for the keys and/or hash functions. This would mean that a secure implementation would require *two* independent RSA key-pairs and up to *three* independent hash functions. The implementation overhead would be quite significant, particularly in terms of code complexity, which could lead to errors that affect security. Additionally, the storage requirements for two independent keys could be prohibitive in certain low resource environments. The common solution is to share the key material and hash function(s) amongst the schemes.

It would be ideal to able to say that since the schemes are both secure then their composition must also be secure, but this not clear in the case where the keys are shared. One might then think that the proofs are somehow composable, but again this is not true. Indeed, as noted in the PKCS#1 standard:

> "Suppose an RSA key pair is employed in both RSASSA-PSS (Section 8.1) and RSASSA-PKCS1-v1_5. Then the security proof for RSASSA-PSS would no longer be sufficient since the proof does not account for the possibility that signatures might be generated with a second scheme." - [RFC 8017, Sec. 6].

In fact there are two distinct problems that arise when trying to naïvely compose the two proofs. Firstly, in the proof by Jager et al. [JKM18a, JKM18b], the modulus N is modified to allow the simulation of signatures, where as the proofs for RSA-PSS do not. This problem can be easily overcome, as the RSA-PSS

[1] See `https://www.ietf.org/mail-archive/web/tls/current/msg19360.html`

proofs are agnostic of the prime factorisation of N. The second, and more critical problem is the sharing of hash functions. Both proofs are in the Random Oracle Model (ROM) and program the oracle responses according to their own needs, which are not compatible with each other and simply combining the proofs would lead to inconsistencies in the oracle responses. This is especially critical given if both schemes employ the Mask Generation Function `MGF1` (cf. [RFC 2437, Sec. 10.2.1]), which uses repeated hash function calls to provide a larger hash output. The reduction needs to take care that these repeated queries do not create inconsistencies.

RSA-PSS has been well studied in the previous years, in particular there have been some improvements upon the original proof by Coron [Cor02a] and Kakvi and Kiltz [KK12, KK18]. However, both these results consider the original scheme as proposed by Bellare and Rogaway [BR96]. While some of the techniques suggested by Coron [Cor02a] were incorporated into Jonsson's proof of the standardized variant of PSS [Jon01], the more recent advances [KK12, KK18] have not.

Given these factors, we find it to be of great interest to study and understand the security of RSA-PSS and PKCS#1 v1.5 in light of these new developments. In particular, we are able to show security when PSS is used with no randomness, making both schemes deterministic. This is quite desirable, as not only can randomness generation be difficult in some scenarios, but it also makes the schemes subversion resistant [AMV15].

1.1 Our Contribution

We present two proofs of security for the joint use of RSA-PSS and RSA-PKCS#1 v1.5 signatures schemes, which we call RSA-TLS-Sibling Signatures. To do this, we first present a new proof for the standardised variant of RSA-PSS, using the techniques of Kakvi and Kiltz [KK12, KK18] for RSA Full Domain Hash (RSA-FDH) [BR93] and the original RSA-PSS [BR96] and give a security proof independent of the amount of randomness used. We then show how one can combine our new proof with that of Jager et al [JKM18a, JKM18b], to give security proofs for the joint use of RSA-PSS and RSA-PKCS#1 v1.5 signatures, again where no randomness is required. We begin with a more idealized proof and then we move toward proving a more practical variant. The first proof covers the "ideal case" where the schemes are implemented with three independent hash functions. We also discuss how this ideal case could be realized in practice. After this, we present a proof that is the closest to reality wherein the schemes are implemented with one shared hash function. Our proofs are tightly secure in the ROM to the lossiness of the RSA permutation, based on the φ-Hiding Assumption (ΦHA) [CMS99]. Both proofs inherit Jager et al.'s parameter expansion [JKM18a, JKM18b]. This is of course to be expected, as it is clear that combining two schemes with disparate security levels cannot result in something that is more secure than the least secure component. It should be noted that all proofs do not depend on the randomness used, and in particular, no randomness can be used, which would make the schemes subversion resistant [AMV15].

It should be noted that all our proofs are tightly secure in the Random Oracle Model. This is possible due to the fact that we rely on the lossiness of the RSA permutation, and not its one-wayness. In particular, this allows to circumvent Coron-style impossibility results [Cor00, Cor02a, BJLS16]. This follows from the fact that RSA is only a certified trapdoor permutation [BY93, BY96] for "large" exponents [KKM12], but we only consider the case where "small" exponents are used.

1.2 Related Work

The first proof for the original RSA-PSS scheme was presented by Bellare and Rogaway [BR96]. This proof was later improved by Coron [Cor02a], who also showed the optimal bounds for the amount of randomness needed. Kakvi and Kiltz [KK12, KK18] later showed that this optimality only applies when the RSA keys define a certified trapdoor permutation [BY96, KKM12], but does not apply for small exponents, where RSA is lossy [PW08]. Based on the lossiness, Kakvi and Kiltz [KK12, KK18] presented a proof for the original RSA-PSS, which was tightly secure with no randomness.

On one hand, the security of the variant used in practice has only been studied by Jonsson [Jon01]. Jonsson reduces the proof of the PSS variant to the original PSS scheme and then further relies on the

original proof of Bellare and Rogaway [BR96]. The proof of Jonsson is very general and presents several parameters that can be selected according to one's needs. This has the unfortunate side effect of making the proof somewhat more cumbersome and difficult to parse.

On the other hand, the only known security proofs for PKCS#1 v1.5 were presented by Jager et al. [JKM18a, JKM18b]. While they were able to achieve a tight reduction to lossiness, there is still a parameter increase. This is due to the fact that Jager et al.'s proof must double the size of the modulus in order to be able to simulate signatures. This means that the proof only holds with specific parameter choices of PKCS#1 v1.5.

It must also be noted that the Rabin variant, with e being fixed to 2, of the PKCS#1 v1.5 and the ISO 9796-2 [ISO 9796-2] signatures were proven to be secure by Coron [Cor02b]. Interestingly enough, this proof has similar restrictions to the one by Jager et al. [JKM18a, JKM18b].

Apart from the aforementioned works, lossiness of the RSA permutation and its applicability to security proofs have been well studied in the literature [KOS10, LOS13, KPS13, Seu14, SZ15].

These types of related signatures are generally not considered in the literature, a first model was proposed by Camenish et al. [CDD17], under the name of *Sibling Signatures*. We will follow their framework and present a proof for joint use of PKCS#1 v1.5 and RSA-PSS signatures.

However, what has been considered in the literature is the use of single key for signing and encryption, formalised by Haber and Pinkas [HP01]. While there have been several works in this direction, including a revisit of the model by Paterson et al. [PSST11], two specific results are quite close to our results in spirit, in that they prove security of an extant real life system. The first is the proof of this for the widely deployed EMV Standard is indeed secure due to Degabriele et al. [DLP+12] and the second is a proof of the TLS handshake by Bhargavan et al. [BFK+14]. It must be noted for the latter result, one also needs the concept of cryptographic agility [ABBC10].

2 Preliminaries

2.1 Notations and conventions

We denote our security parameter as $\lambda \in \mathbb{N}$. For all $n \in \mathbb{N}$, we denote by 0^n and 1^n the n-bit string of all zeroes and all ones, respectively. We denote the concatenation of two bitstrings x and y as $x||y$. For any set S, we use $x \in_R S$ to indicate that we choose x uniformly random from S. All algorithms may be randomized. For any algorithm A, we define $x \leftarrow_\$ A(a_1, \ldots, a_n)$ as the execution of A with inputs $a_1, \ldots, a_n$ and fresh randomness and then assigning the output to x. For deterministic algorithms, we drop the \$ from the arrow. We denote the set of prime numbers by $\mathbb{P}$ and we denote the subset of κ-bit primes as $\mathbb{P}[\kappa]$. Similarly, we denote the set of κ-bit integers as $\mathbb{Z}[\kappa]$. We denote by $\mathbb{Z}_N^*$ the multiplicative group modulo $N \in \mathbb{N}$. For any $a, b \in \mathbb{Z}$, with $a < b$ we denote the set $\{a, a+1, \ldots, b-1, b\}$ with $[\![a, b]\!]$. For any $n \in \mathbb{N}$ and for any $a \in \mathbb{N}[\kappa]$, with $\kappa < n$, we denote by $\langle a \rangle_n$ the binary representation of a padded to n bits, i.e. $\langle a \rangle_n = 0^{n-\kappa}||a$. For any bit string x of sufficient length, we denote by $\texttt{MSBs}(x, n)$ the n most significant (leading) bits of x and $\texttt{LSBs}(x, n)$ the n least significant (trailing) bits of x. We will use game based proofs and will denote by $\mathsf{G}^{\mathcal{A}} \Rightarrow 1$ the event that the adversary $\mathcal{A}$ wins game G. All code that is only in specific games with be indicated with a comment preceded by two forward slashes e.g. **//New code for G_1**.

2.2 Signature Schemes

We first recall the standard definition of a signature scheme, as well as its security.

Definition 1. A digital signature scheme $\texttt{Sig}$ with message space $\mathbb{M}$ and signature space $\mathbb{S}$ is defined as a triple of probabilistic polynomial time (PPT) algorithms $\texttt{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$:

- KeyGen takes as an input the unary representation of our security parameter 1^λ and outputs a signing key sk and verification key pk.

Game UF-CMA(ROM)

Initialize(1^λ)
$(\mathsf{pk}, \mathsf{sk}) \leftarrow_\$ \mathtt{Sig}.\mathsf{KeyGen}(1^\lambda)$
return pk

Hash(m)
if $(m, \cdot) \in \mathcal{H}$
    fetch $(m, y) \in \mathcal{H}$
    return y
else
    $y \in_R \{0,1\}^\ell$;
    $\mathcal{H} \leftarrow \mathcal{H} \cup \{(m, y)\}$
    return y

Sign(m)
$\mathcal{M} \leftarrow \mathcal{M} \cup \{m\}$
return $\sigma \leftarrow_\$ \mathtt{Sig}.\mathsf{Sign}(sk, m)$

Finalize(m^*, σ^*)
if $\mathtt{Sig}.\mathsf{Verify}(\mathsf{pk}, m^*, \sigma^*) == 1 \wedge m^* \notin \mathcal{M}$
    return 1
else
    return 0

Figure 1: UF-CMA security game in the Random Oracle Model

- Sign takes as input a signing key sk, message $m \in \mathbb{M}$ and outputs a signature $\sigma \in \mathbb{S}$.
- Verify is a deterministic algorithm, which on input of a public key and a message-signature pair $(m, \sigma) \in \mathbb{M} \times \mathbb{S}$ outputs 1 (accept) or 0 (reject).

We say $\mathtt{Sig}$ is correct if for any $\lambda \in \mathbb{N}$, all $(\mathsf{pk}, \mathsf{sk}) \leftarrow_\$ \mathsf{KeyGen}(1^\lambda)$, all $m \in \mathbb{M}$, and all $\sigma \leftarrow_\$ \mathsf{Sign}(\mathsf{sk}, m)$ we have that

$$\Pr[\mathsf{Verify}(\mathsf{pk}, m, \sigma) = 1] = 1.$$

For signature security, we consider the notion of *UnForgeability under adaptive Chosen Message Attack in the Random Oracle Model* (UF-CMA(ROM)). The security experiment is presented in Figure 1. It must be noted that all hash function calls are replaced with a call the the random oracle. In the case where we have multiple hash functions, we have multiple hash functions.

We say that $\mathtt{Sig}$ is $(t, \varepsilon, q_h, q_s)$-UF-CMA(ROM) secure if for any forger $\mathcal{F}$ running in time at most t, making at most q_h hash queries and making at most q_s signature queries, we have:

$$\mathbf{Adv}^{\mathsf{UF\text{-}CMA(ROM)}}_{\mathcal{F},\mathtt{Sig}} = \Pr\left[\begin{matrix} 1 \leftarrow \mathbf{Finalize}(m^*, \sigma^*); \\ (m^*, \sigma^*) \leftarrow \mathcal{F}^{\mathbf{Hash}(\cdot),\mathbf{Sign}(\cdot)}(\mathsf{pk}) \end{matrix}\right] \leqslant \varepsilon$$

We now recall the definition of a sibling signature scheme due Camenisch et al. [CDD17].

Definition 2. A sibling signature scheme $\mathtt{SibSig}$ with message spaces $\mathbb{M}_1, \mathbb{M}_2$ and signature spaces $\mathbb{S}_1, \mathbb{S}_2$ is defined as a hextuple of probabilistic polynomial time (PPT) algorithms $\mathtt{SibSig} = (\mathsf{SibSetup}, \mathsf{SibKeyGen}, \mathsf{Sign}_1, \mathsf{Sign}_2, \mathsf{Verify}_1, \mathsf{Verify}_2)$:

- $\mathsf{SibSetup}$ takes as an input the unary representation of our security parameter 1^λ and outputs the system parameters sp.

- KeyGen takes as input the system parameters sp and outputs the signing key sk and verification key pk.
- Sign_1 takes as input the signing key sk, message $m \in \mathbb{M}_1$ and outputs a signature $\sigma \in \mathbb{S}_1$.
- Sign_2 takes as input the signing key sk, message $m \in \mathbb{M}_2$ and outputs a signature $\sigma \in \mathbb{S}_2$.
- Verify_1 is a deterministic algorithm, which on input of the public key pk and a message-signature pair $(m, \sigma) \in \mathbb{M}_1 \times \mathbb{S}_1$ outputs 1 (accept) or 0 (reject).
- Verify_2 is a deterministic algorithm, which on input of the public key pk and a message-signature pair $(m, \sigma) \in \mathbb{M}_2 \times \mathbb{S}_2$ outputs 1 (accept) or 0 (reject).

We say $\mathtt{SibSig}$ is correct if for $i \in \{1, 2\}$, for all $\lambda \in \mathbb{N}$, all key pairs $(\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{sk}_1, \mathsf{sk}_2) \leftarrow_{\$} \mathsf{KeyGen}(1^\lambda)$, all message $m \in \mathbb{M}_i$, and all $\sigma \leftarrow_{\$} \mathsf{Sign}_i(\mathsf{sk}_i, m)$ we have that

$$\Pr[\mathsf{Verify}_i(\mathsf{pk}_i, m, \sigma) = 1] = 1.$$

We now recall the security of Sibling Signatures. While it is never given an explicit name by Camenisch et al.[CDD17], which we call *SIBling UnForgeability under Chosen Message Attack in the Random Oracle Model* ($\mathsf{SIB\text{-}UF(ROM)}$) in Figure 2.

We say that $\mathtt{SibSig}$ is $(t, \varepsilon, q_h, q_{s_1}, q_{s_2})$-$\mathsf{SIB\text{-}UF(ROM)}$ secure if for any forger $\mathcal{F}$ running in time at most t, making at most q_h hash queries, making at most q_{s_1} signature queries to Sign_1 and making at most q_{s_2} signature queries to Sign_2, we have:

$$\mathbf{Adv}^{\mathsf{SIB\text{-}UF(ROM)}}_{\mathcal{F},\mathtt{SibSig}} = \Pr\left[\begin{matrix} 1 \leftarrow \mathbf{Finalize}(m^*, \sigma^*); \\ (m^*, \sigma^*) \leftarrow \mathcal{F}^{\mathbf{Hash}(\cdot),\mathbf{Sign_1}(\cdot),\mathbf{Sign_2}(\cdot)}(\mathsf{pk}_1, \mathsf{pk}_2) \end{matrix}\right] \leqslant \varepsilon$$

2.3 Computational Assumptions

We now recall the φ-Hiding Assumption by Cachin et al. [CMS99], which we denote as $\Phi\mathsf{HA}[\lambda]$[2]. The $\Phi\mathsf{HA}[\lambda]$ essentially states that given a modulus N and a sufficiently small exponent e, it is hard to decide if $e|\varphi(N)$ or not. In this case sufficiently small means that $e < N^{\frac{1}{4}}$, as for larger exponents, Kakvi et al. [KKM12] show how to decide this using Coppersmith's method [Cop97].

Assumption 1 (The φ-Hiding Assumption. [CMS99])**.** The φ-Hiding Assumption, denoted by $\Phi\mathsf{HA}[\lambda]$, states that it is hard to distinguish between $(N, \mathfrak{e}_{inj})$ and $(N, \mathfrak{e}_{los})$, where N is the product of 2 random $(\lambda/2)$-bit primes and $\mathfrak{e}_{inj}, \mathfrak{e}_{los} > 3 \in \mathbb{P}$ and and $\mathfrak{e}_{inj}, \mathfrak{e}_{los}, \leq N^{\frac{1}{4}}$, with $gcd(\mathfrak{e}_{inj}, \varphi(N)) = 1$ and $gcd(\mathfrak{e}_{los}, \varphi(N)) = \mathfrak{e}_{los}$, where φ is the Euler Totient function. $\Phi\mathsf{HA}[\lambda]$ is said to be (t, ε)-hard, if for all distinguishers $\mathcal{D}$ running in time at most t, we have:

$$\mathbf{Adv}^{\Phi\mathsf{HA}[\lambda]}_{\mathcal{D}} = \Pr\left[1 \leftarrow \mathcal{D}(N, \mathfrak{e}_{inj})\right] - \Pr\left[1 \leftarrow \mathcal{D}(N, \mathfrak{e}_{los})\right] \leqslant \varepsilon$$

Note that when $\gcd(\mathfrak{e}_{los}, \varphi(N)) = \mathfrak{e}_{los}$, the function $x^{\mathfrak{e}_{los}} \bmod N$ is exactly $\mathfrak{e}_{los}$-to-1, i.e. it said to be $\mathfrak{e}_{los}$*-regular lossy* as defined by Kakvi and Kiltz [KK12, KK18].

3 The Probabilistic Signature Scheme

We now begin by recalling the RSA-Probabilistic Signature Scheme (RSA-PSS) as defined by Bellare and Rogaway [BR96], which we will refer to as $\mathtt{PSS96}$, for completeness. $\mathtt{PSS96}$ is paramaterized by two integers $\ell_{\mathtt{R}}$ and $\ell_{\mathtt{H}}$. The value $\ell_{\mathtt{R}}$ determines the size of the random salt, that is to say our randomness space is $\{0,1\}^{\ell_{\mathtt{R}}}$. $\ell_{\mathtt{H}}$ defines the output size of our first hash function $\mathtt{H} : \mathbb{M} \to \{0,1\}^{\ell_{\mathtt{H}}}$, which implicitly determines

[2]We explicitly include the security parameter for clarity in the reductions.

Game $\mathsf{SIB\text{-}UF}(\mathsf{ROM})$

procedure Initialize(1^λ)
$\mathsf{sp} \leftarrow_\$ \mathtt{SibSig}.\mathsf{SibSetup}(1^\lambda)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow_\$ \mathtt{SibSig}.\mathsf{KeyGen}(\mathsf{sp})$
return (pk)

procedure Hash(m)
if $(m, \cdot) \in \mathcal{H}$
    fetch $(m, y) \in \mathcal{H}$
    return y
else
    $y \in_R \mathsf{Domain}$;
    $\mathcal{H} \leftarrow \mathcal{H} \cup \{(m, y)\}$
    return y

procedure Sign$_1(m)$
$\mathcal{M}_1 \leftarrow \mathcal{M}_1 \cup \{m\}$
return $\sigma \leftarrow_\$ \mathsf{Sign}_1(\mathsf{sk}, m)$

procedure Sign$_2(m)$
$\mathcal{M}_2 \leftarrow \mathcal{M}_2 \cup \{m\}$
return $\sigma \leftarrow_\$ \mathsf{Sign}_2(\mathsf{sk}, m)$

procedure Finalize(m^*, σ^*, i^*)
if $(i^* == 1 \wedge m^* \notin \mathcal{M}_1)$
    if $\mathsf{Verify}_1(\mathsf{pk}, m^*, \sigma^*) == 1$
        return 1
if $(i^* == 2 \wedge m^* \notin \mathcal{M}_2)$
    if $\mathsf{Verify}_2(\mathsf{pk}, m^*, \sigma^*) == 1$
        return 1
else
    return 0

Figure 2: SIB-UF-CMA Security Game in the Random Oracle Model

Scheme $\texttt{PSS96}[k, \ell_{\texttt{R}}, \ell_{\texttt{H}}]$ [BR96]

algorithm KeyGen(1^λ)
$N = 1, \varphi(N) = 1$
for $i \in [\![1, k]\!]$
  $p_i \in_R \mathbb{P}[\lambda/k]$
  $N = N \cdot p_i$
  $\varphi(N) = \varphi(N) \cdot (p_i - 1)$
end for
$e \in_R \mathbb{Z}_N^*, \gcd(e, \varphi(N)) = 1$
pick hash functions
  $\texttt{H} : \mathbb{M} \to \{0,1\}^{\ell_{\texttt{H}}}$
  $\texttt{G} : \{0,1\}^{\ell_{\texttt{H}}} \to \{0,1\}^{\ell_{\texttt{G}} = \lambda - \ell_{\texttt{H}} - 1}$
return $(pk = (N, e, \texttt{H}, \texttt{G}), \texttt{sk} = (p_1, \ldots, p_k))$

algorithm Sign$(\texttt{sk}, m)$
$r \in_R \{0,1\}^{\ell_{\texttt{R}}}$
$\omega \leftarrow \texttt{H}(m||r)$
$r^* \leftarrow \texttt{G}_1(\omega) \oplus r$
$y = 0||\omega||r^*||\texttt{G}_2(\omega)$
return $\sigma = y^{1/e} \mod N$

algorithm Verify$(\texttt{pk}, m, \sigma)$
$y = \sigma^e \mod N$
parse y as $0||\omega||r^*||\gamma$
$r = r^* \oplus \texttt{G}_1(\omega)$
if $(\texttt{H}(m||r) == \omega \wedge \texttt{G}_2(\omega) == \gamma)$
  return 1
else
  return 0

(a) The RSA Probabilistic Signature Scheme PSS96

Scheme $\texttt{PSS00}[k, \ell_{\texttt{R}}, \ell_{\texttt{H}}]$ [IEEE 1363]

algorithm KeyGen(1^λ)
$N = 1, \varphi(N) = 1$
for $i \in [\![1, k]\!]$
  $p_i \in_R \mathbb{P}[\lambda/k]$
  $N = N \cdot p_i$
  $\varphi(N) = \varphi(N) \cdot (p_i - 1)$
end for
$e \in_R \mathbb{Z}_N^*, \gcd(e, \varphi(N)) = 1$
pick hash functions
  $\texttt{H} : \mathbb{M} \to \{0,1\}^{\ell_{\texttt{H}}}$
  $\texttt{G} : \{0,1\}^{\ell_{\texttt{H}}} \to \{0,1\}^{\ell_{\texttt{G}} = \lambda - \ell_{\texttt{H}} - 9}$
return $(pk = (N, e, \texttt{H}, \texttt{G}), \texttt{sk} = (p_1, \ldots, p_k))$

algorithm Sign$(\texttt{sk}, m)$
$\mu = \texttt{H}(m)$
$r \in_R \{0,1\}^{\ell_{\texttt{R}}}$
$\omega \leftarrow \texttt{H}(0^{64}||\mu||r)$
$r^* \leftarrow \texttt{G}_2(\omega) \oplus 1||r$
$y = 0||\texttt{G}_1(\omega)||r^*||\omega||10111100$
return $\sigma = y^{1/e} \mod N$

algorithm Verify$(\texttt{pk}, m, \sigma)$
$y = \sigma^e \mod N$
parse y as $0||\gamma||r^*||\omega||10111100$
$1||r = r^* \oplus \texttt{G}_2(\omega)$
if $(\texttt{H}(0^{64}||\texttt{H}(m)||r) == \omega \wedge \texttt{G}_1(\omega) == \gamma)$
  return 1
else
  return 0

(b) The RSA Probabilistic Signature Scheme PSS00.

Figure 3: The two version of the RSA-Probabilistic Signature Scheme RSA-PSS.

the input and output size of the second hash function $\texttt{G} : \{0,1\}^{\ell_{\texttt{H}}} \to \{0,1\}^{\lambda - \ell_{\texttt{H}} - 1}$. For consistency, we define $\ell_{\texttt{G}} = \lambda - \ell_{\texttt{H}} - 1$. For simplicity, we define $\texttt{G}_1(x) = \texttt{MSBs}(\texttt{G}(x), \ell_{\texttt{R}})$ and $\texttt{G}_2(x) = \texttt{LSBs}(\texttt{G}(x), \ell_{\texttt{G}} - \ell_{\texttt{R}})$. We extend PSS96 to allow multi-prime moduli and we denotes the number of primes by k. We will use the notation $\texttt{PSS96}[k, \ell_{\texttt{R}}, \ell_{\texttt{H}}]$ to define the scheme with those parameters. We recall the scheme in Figure 3a.

Now we recall the modified variant of PSS96 that was standardised by the IEEE in the P1363-2000 standard [IEEE 1363] which we will refer to as PSS00. The changes from PSS96 to PSS00 are documented in [Jon01] and briefly recalled in Section 3.1. While PSS00 has the same parameters are PSS96, there are some differences. Firstly, we have that $\ell_{\texttt{G}}$ which is now defined as $\ell_{\texttt{G}} = \lambda - \ell_{\texttt{H}} - 9$. This is due to the 8 additional fixed bits that are appended after the hash function. Additionally, due to the changes in the order of hash functions, we have $\texttt{G}_2(x) = \texttt{LSBs}(\texttt{G}(x), \ell_{\texttt{R}} + 1)$ and $\texttt{G}_1(x) = \texttt{MSBs}(\texttt{G}(x), \ell_{\texttt{G}} - \ell_{\texttt{R}} - 1)$. We will use the notation $\texttt{PSS00}[k, \ell_{\texttt{R}}, \ell_{\texttt{H}}]$ to define the scheme with those parameters. We recall the scheme in Figure 3b.

3.1 Transforming PSS96 to PSS00

We now discuss the changes made to PSS96 and their effect on the security proof. We stress that the order in which we present the changes is not representative of the order in which the changes were made, but rather

the order in which thought would be simplest to explain. If we write out the PSS96 signing algorithm, we get:

$$\mathsf{Sign}(\mathsf{sk}, m; r) = (0||\mathtt{H}(m||r)||(\mathtt{G}(\mathtt{H}(m||r)) \oplus (r||0\ldots0)))^{1/e} \bmod N$$

The first change that is made is that G and H are swapped around. This change is quite small and only requires a superficial change to the security proof. Our signing equation is now:

$$\mathsf{Sign}(\mathsf{sk}, m; r) = (0||(\mathtt{G}(\mathtt{H}(m||r)) \oplus (r||0\ldots0))||\mathtt{H}(m||r))^{1/e} \bmod N$$

The next change is that we now XOR our randomness with $\mathtt{G}_2$ instead of $\mathtt{G}_1$. As with the previous change, only a superficial change is needed to the security proof. This gives the signing equation:

$$\mathsf{Sign}(\mathsf{sk}, m; r) = (0||(\mathtt{G}(\mathtt{H}(m||r)) \oplus (0\ldots0||r))||\mathtt{H}(m||r))^{1/e} \bmod N$$

The last change to our randomness is that we add a 1-bit delimiter before the randomness. Again, the proof can be adapted with a minor change. This means the signing equation is:

$$\mathsf{Sign}(\mathsf{sk}, m; r) = (0||(\mathtt{G}(\mathtt{H}(m||r)) \oplus (0\ldots01||r))||\mathtt{H}(m||r))^{1/e} \bmod N$$

The next change is that we no longer hash the message, but a hash of the message. This change is not quite as drastic as it first seems. The fact that we sign hashes of fixed-length inputs means we can distinguish between queries to our oracle for signing vs. other queries, which actually helps us in our proof. The changes required to fix this are more involved when we add all our other changes. Thus, our signing equation becomes:

$$\mathsf{Sign}(\mathsf{sk}, m; r) = (0||(\mathtt{G}(\mathtt{H}(\mathtt{H}(m)||r)) \oplus (0\ldots01||r))||\mathtt{H}(\mathtt{H}(m)||r))^{1/e} \bmod N$$

The next change is for the option to use message recovery. We additionally encode the length of the message to be recovered into the value being hashed. The current standard specifies a length field of length 64 bits. If there is no message to recover, the length is set to 0. This change, in combination with the other changes, is where the proof starts to become more complicated. The change required here requires careful treatment of all the cases. Concretely, the signing equation is:

$$\mathsf{Sign}(\mathsf{sk}, m; r) = (0||(\mathtt{G}(\mathtt{H}(0^{64}||\mathtt{H}(m)||r)) \oplus (0\ldots01||r))||\mathtt{H}(0^{64}||\mathtt{H}(m)||r))^{1/e} \bmod N$$

The final change is for compatibility reasons. In order to be compatible with the Integer Factorization Signature Primitive using Rabin-Williams (IFSP-RW) [IEEE 1363, Sec 8.2.8], a 0xbc is concatenated to the end of the hash. This now has an interesting effect on the proof, as we now have to resample until we have a trailing 0xbc. This increases the running time of our reduction by an expected factor of 2^8, which means we lose 8 bits of security. This gives us our final signing equation of:

$$\mathsf{Sign}(\mathsf{sk}, m; r) = (0||(\mathtt{G}(\mathtt{H}(0^{64}||\mathtt{H}(m)||r)) \oplus (0\ldots01||r))||\mathtt{H}(0^{64}||\mathtt{H}(m)||r)||\mathtt{0xbc})^{1/e} \bmod N$$

This security loss comes from the work factor of our reductions. The work factor ($\mathcal{WF}$) of an adversary $\mathcal{A}$ is defined as:

$$\mathcal{WF} = \frac{\varepsilon_{\mathcal{A}}}{t_{\mathcal{A}}}$$

When we say we have κ-bit security, this translates to having a work factor of at least 2^κ. Since the reduction for PSS96 and PSS00 have the same advantage, but the time differs by 2^8, we have a loss of 7 bits of security. Suppose for some parameters we have a work factor of 2^κ, i.e. κ bits of security, for PSS96, then we have:

$$\frac{\varepsilon_{\mathtt{PSS00}}}{t_{\mathtt{PSS00}}} = \frac{\varepsilon_{\mathtt{PSS96}}}{2^8 \cdot t_{\mathtt{PSS96}}} = \frac{1}{2^8} \cdot \frac{\varepsilon_{\mathtt{PSS96}}}{t_{\mathtt{PSS96}}} \frac{2^\kappa}{2^8} = 2^{\kappa-8}$$

4 New Security Proofs for PSS

We now present our new proofs for the security of PSS00, following the lossy proof methodology of Kakvi and Kiltz [KK12, KK18] and proving a result in a similar vein. Unlike the previous proof [Jon01], our proof is independent of the size of the random salt ℓ_{R}. This means in particular, we can set $\ell_{\mathsf{R}} = 0$ and use PSS00 in a deterministic manner without any loss in security. Furthermore, setting $\ell_{\mathsf{R}} > 0$ does not have any security gain. We now present our proof, which follows the template of the proof of Kakvi and Kiltz [KK12, KK18]. First we show proofs for the case where we use two hash functions and then we show how one can use one hash function and give a proof for this.

4.1 Proof with 2 Hash Functions

Theorem 1. *Assume RSA is a regular (η, t', ε')-lossy trapdoor permutation for $\eta \geq 2$. Then, for any $(q_h, q_g, q_s, \ell_{\mathsf{R}}, \ell_{\mathtt{H}}), \mathsf{PSS00}[2, \lambda, \ell_{\mathsf{R}}, \ell_{\mathtt{H}}]$ is $(t, \varepsilon, q_h, q_g, q_s)$-UF-CMA secure in the Random Oracle Model, where*

$$\begin{aligned} \varepsilon &\leqslant \left(\frac{2\eta - 1}{\eta - 1}\right) \cdot \varepsilon' + \frac{(q_h + q_s)^2}{2^{\ell_{\mathtt{H}}}} \\ t' &\approx t + 2^8 \cdot (q_h + q_s) \cdot t_{\mathsf{RSA}}, \end{aligned}$$

and t_{RSA} is the time to compute one modular exponentiation.

Proof. We prove our games through a series of game hops, which we describe in Figures 4a and 4b. We move from our standard EUF-CMA-ROM game to a game where the adversary has an advantage of 0. We show that each transition from game to game is negligible via Lemmas 1-5

The first change we make is to move to Game G_1, where we no longer generate the public key and signing key, but we use an *injective* $\Phi\mathsf{HA}[\lambda]$ instance as our public key and we program our random oracles to simulate signatures. This works, unless we get a collision in one of our hash functions, which happens with a small probability. This is captured by the following lemma.

Lemma 1. $\Pr[\mathsf{G}_1^{\mathcal{F}} \Rightarrow 1] = \Pr[\mathsf{G}_0^{\mathcal{F}} \Rightarrow 1] - \frac{(q_h+q_s)^2}{2^{\ell_{\mathtt{H}}}}$.

Proof. In G_1 we modify the H oracle and the signing oracle. First we distinguish between "signable" queries and "non-signable" queries. Only queries of the form $0^{64}||\mathtt{H}(m)||r \in \{0,1\}^{64+\ell_{\mathtt{H}}+\ell_{\mathtt{R}}}$ will be signed. Thus on these queries, the H oracle now works by sampling a random element $\sigma_{(m,r)} \in \mathbb{Z}_N^*$ and raises computes a candidate message representative $y = \sigma_{(m,r)}^e \bmod N$. This process is repeated until the trailing 16 bits of y are equal to 10111100[3] This takes an expected 2^8 resamplings. Once a valid message representative y is found, it is parsed as $0||\gamma_{(m,r)}||r^*_{(m,r)}||\omega_{(m,r)}||10111100$. We now check if $\omega_{(m,r)}$ has been previously queried to the G oracle. If it has, then we abort, as this would lead to an inconsistent oracle state. If this is not the case, we program the G oracles to return this value. The H oracle the stores the response $\omega_{(m,r)}$ and the signature $\sigma_{(m,r)}$. The signing oracle makes the requisite oracle calls, to ensure that all values are defined and then simply retrieves the signature and returns $\sigma_{(m,r)}$. It is clear to see that this indeed a valid signature. Since we are using the injective variant, the RSA function is a permutation, hence the response of H oracle is also correctly distributed. Hence, the simulation of signatures is correct, unless the reduction aborts. The probability of a collision is at most $\frac{(q_s+q_h)^2}{2^{\ell_{\mathtt{H}}}}$, since the adversary makes at most $q_h + q_s$ queries, implicit or explicit, to the G-oracle, giving at most $(q_h+q_s+1)^2$ possible collisions from a total of $2^{\ell_{\mathtt{H}}}$ possible choices. Furthermore, we do not abort if collision matches the value already set, which happens with a probability $1 - \frac{1}{2^{\lambda-\ell_{\mathtt{H}}}}$. Hence the total probability that we abort is $\left(1 - \frac{1}{2^{\lambda-\ell_{\mathtt{H}}}}\right)\left(\frac{(q_h+q_s+1)^2}{2^{\ell_{\mathtt{H}}}}\right)$. Thus we have $\Pr[\mathsf{G}_1^{\mathcal{F}} \Rightarrow 1] = \Pr[\mathsf{G}_0^{\mathcal{F}} \Rightarrow 1] - \frac{(q_h+q_s+1)^2}{2^{\ell_{\mathtt{H}}}}$. □

[3] 10111100 is the binary for 0xbc as specified in the standard [IEEE 1363].

Game G_0

Initialize(1^λ)
$(\mathsf{pk}, \mathsf{sk}) \leftarrow_\$ \mathtt{PSS00}[2, \lambda, \ell_\mathsf{R}, \ell_\mathtt{H}].\mathsf{KeyGen}(1^\lambda)$
return pk

Oracle H(x)
if $(x, \cdot) \in \mathcal{H}$
  fetch $(x, \omega_x) \in \mathcal{H}$
  return ω_x
else
  $\omega_x \in_R \{0,1\}^{\ell_\mathtt{H}}$
  set $\mathcal{H}[x] = (\omega_{(m,r)}, \sigma_{(m,r)})$
  return ω_x

Oracle G(x)
if $(x, \cdot) \in \mathcal{G}$
  fetch $(x, \alpha_x) \in \mathcal{G}$
  return α_x
else
  $\alpha_x \in_R \{0,1\}^{\ell_\mathtt{G}}$
  set $\mathcal{G}[x] = (\alpha_x)$
  return α_x

Sign(m)
$\mathcal{M} \leftarrow \mathcal{M} \cup \{m\}$
$\sigma \leftarrow_\$ \mathtt{PSS00}[2, \ell_\mathsf{R}, \ell_\mathtt{H}].\mathsf{Sign}(\mathsf{sk}, m)$
return σ

Finalize(m^*, σ^*)
if $(m^* \in \mathcal{M})$
  return 0
if $(\mathtt{PSS00}[2, \ell_\mathsf{R}, \ell_\mathtt{H}].\mathsf{Verify}(\mathsf{pk}, m^*, \sigma^*) == 1))$
  return 1
else
  return 0

(a) Game G_0 for Theorem 1

Games $\mathsf{G}_1 - \mathsf{G}_4$

Initialize(1^λ)
$(N, \mathfrak{e}_{inj}, \mathfrak{e}_{los}) \leftarrow_\$ \Phi\mathsf{HA}[\lambda]$
$e = \mathfrak{e}_{inj}$ **//Injective keys for $\mathsf{G}_1, \mathsf{G}_4$**
$e = \mathfrak{e}_{los}$ **//Lossy keys for $\mathsf{G}_2, \mathsf{G}_3$**
return $\mathsf{pk} = (N, \hat{e})$

Oracle H(x)
if $(\mathcal{H}[x]$ is defined)
  fetch $\mathcal{H}[x] = (\omega_x, \sigma_x)$
  return ω_x
else if $(|x| = \ell_\mathtt{H} + \ell_\mathsf{R} + 64$ and $\mathtt{MSBs}(x, 64) = 0^{64})$
  parse $x = 0^{64}||m||r$
  repeat
    $\sigma_{(m,r)} \in_R \mathbb{Z}_N$
    $y_{(m,r)} = \sigma_{(m,r)}^e \bmod N$
  until $\mathtt{LSBs}(y_{(m,r)}, 16) = 10111100$
  Parse $y_{(m,r)} = 0||\gamma_{(m,r)}||r^*_{(m,r)}||\omega_{(m,r)}||10111100$
  if $(\mathcal{G}[\omega_{(m,r)}]$ is defined)
    return $\perp$
  $\alpha_{m,r} = \gamma_{m,r}||(r^*_{(m,r)} \oplus 1||r)$
  set $\mathcal{G}[\omega_{(m,r)}] = (\alpha_{(m,r)})$
  set $\mathcal{H}[x] = (\omega_{(m,r)}, \sigma_{(m,r)})$
  return $\omega_{(m,r)}$
else
  $\omega_x \in_R \{0,1\}^{\ell_\mathtt{H}}$
  set $\mathcal{H}[x] = (\omega_x, \perp)$
  return ω_x

Oracle G(x)
if $(x, \cdot) \in \mathcal{G}$
  fetch $\mathcal{G}[x] = (\alpha_x)$
  return α_x
else
  $\alpha_x \in_R \{0,1\}^{\ell_\mathtt{G}}$
  set $\mathcal{G}[x] = (\alpha_x)$
  return α_x

Sign(m)
$\mathcal{M} \leftarrow \mathcal{M} \cup \{m\}$
call $\mathtt{H}(m)$
fetch $\mathcal{H}[m] = (\omega_m, \sigma_m)$
$r \in_R \{0,1\}^{\ell_\mathsf{R}}$
call $\mathtt{H}(0^{64}||\omega_m||r)$
fetch $\mathcal{H}[0^{64}||\omega_m||r] = (\omega_m, \sigma_{(m,r)})$
return $\sigma_{m,r}$

Finalize(m^*, σ^*)
if $m^* \in \mathcal{M}$
  return 0
$y = (\sigma^*)^e \bmod N$
if$(\mathtt{LSBs}(y, 16)! = 10111100)$
  return 0
parse $y = 0||\gamma||r^*||\omega||10111100$
$1||r = r^* \oplus \mathtt{LSBs}(\alpha_{\omega^*}, \ell_\mathsf{R} + 1)$
fetch $\mathcal{H}[m^*] = (\omega_{m^*}), \sigma_{m^*})$
fetch $\mathcal{H}[0^{64}||\omega_{m^*}||r] = (\omega^*, \sigma_{(m,r)})$
if $(\sigma^* = \sigma_{(m,r)}$ then **//Abort rule in $\mathsf{G}_3, \mathsf{G}_4$**
  return 0 **//Abort rule in $\mathsf{G}_3, \mathsf{G}_4$**
fetch $\mathcal{G}[\omega^*] = (\alpha_{\omega^*})$
if $(\omega^* == \omega \wedge \mathtt{MSBs}(\alpha_{\omega^*}, \ell_\mathtt{G} - \ell_\mathsf{R} - 1) == \gamma)$
  return 1
else
  return 0

(b) Games $\mathsf{G}_1 - \mathsf{G}_4$ for Theorem 1

Figure 4: Games $\mathsf{G}_0 - \mathsf{G}_4$ for proof of PSS00 with 2 Random Oracles.

The next change to game G_2, where we switch from an *injective* $\Phi\mathsf{HA}[\lambda]$ instance to a *lossy* $\Phi\mathsf{HA}[\lambda]$ instance. This change should not be noticed by the forger. If the forger does notice this change, then we can then use them to construct a distinguisher against the $\Phi\mathsf{HA}[\lambda]$.

Lemma 2. *There exists a distinguisher $\mathcal{D}_1$ against the $\Phi\mathsf{HA}[\lambda]$, which runs in time $t' = t + 2^8 \cdot q_h \cdot t_{\mathsf{RSA}}$ and such that $\left|\Pr[\mathsf{G}_1^{\mathcal{F}} \Rightarrow 1] - \Pr[\mathsf{G}_2^{\mathcal{F}} \Rightarrow 1]\right| = \mathbf{Adv}_{\mathcal{D}_1}^{\Phi\mathsf{HA}[\lambda]}$.*

Proof. From G_1 to G_2, we change the key generation from a normal key (N, e) to a lossy key $(N, \hat{e})$, however the oracles are identical in both games. We now build a distinguisher $\mathcal{D}_1$ against the $\Phi\mathsf{HA}[\lambda]$, using these games. The distinguisher will run $\mathcal{F}$ and simulates the oracles $\mathtt{H}, \mathtt{G}, \mathsf{Sign}$ as described in games G_1 and G_2. Note that $\mathcal{D}_1$ does not require the signing key to simulate the oracles. After $\mathcal{F}$ calls **Finalize**, $\mathcal{D}_1$ returns the output of **Finalize**. Thus we can see that $\Pr[1 \leftarrow \mathcal{D}_1(N, e)] = \Pr\left[\mathsf{G}_1^{\mathcal{F}} \Rightarrow 1\right]$ and $\Pr[1 \leftarrow \mathcal{D}_1(N, \hat{e})] = \Pr\left[\mathsf{G}_2^{\mathcal{F}} \Rightarrow 1\right]$ Hence we have $\left|\Pr\left[\mathsf{G}_1^{\mathcal{F}} \Rightarrow 1\right] - \Pr\left[\mathsf{G}_2^{\mathcal{F}} \Rightarrow 1\right]\right| = |\Pr\left[1 \leftarrow \mathcal{D}_1(N, e)\right] - \Pr\left[1 \leftarrow \mathcal{D}_1(N, \hat{e})\right]| = \mathbf{Adv}_{\mathcal{D}_1}^{\Phi\mathsf{HA}[\lambda]}$. □

We now move to Game G_3, where we add an additional abort condition to Finalize procedure. Here we reject any forgery that is the same as our simulated signature. Since we now have lossy signatures, there are many valid signatures, thus the adversary may submit a forgery that is different from our simulated signature. We show this in the following lemma.

Lemma 3. $\Pr[\mathsf{G}_3^{\mathcal{F}} \Rightarrow 1] = \left(\frac{\eta - 1}{\eta}\right) \Pr[\mathsf{G}_2^{\mathcal{F}} \Rightarrow 1]$.

Proof. In G_3, we an additional abort condition, the reduction if the forgery σ^* provided by $\mathcal{F}$ is the same as the simulated signature $\sigma_{(m^*, r)}$ for the target message m^*, with the randomness r. If this is the case, the reduction outputs 0 and terminates. Notice that the simulated signature $\sigma_{(m^*, r)}$ is independent of $\mathcal{F}$'s view and is uniformly distributed in the set of valid signatures for m^* under randomness r. Due to the fact that the function $x^{\hat{e}} \bmod N$ is η-regular lossy, with $\eta = \hat{e}$, the probability of a collision is $1/\eta$. Thus we see that this abort condition reduces the probability of the forger winning the game by $1/\eta$, hence $\Pr\left[\mathsf{G}_3^{\mathcal{F}} \Rightarrow 1\right] = \left(1 - \frac{1}{\eta}\right) \Pr\left[\mathsf{G}_2^{\mathcal{F}} \Rightarrow 1\right] = \left(\frac{\eta - 1}{\eta}\right) \Pr\left[\mathsf{G}_2^{\mathcal{F}} \Rightarrow 1\right]$. □

The last step in the proof is to move to Game G_4. Here we switch back to an *injective* $\Phi\mathsf{HA}[\lambda]$ instance, which makes our RSA function once again a permutation. As with the change from G_1 to G_2, this is unnoticed by the forger, or we can use the forger to build a distinguisher. Furthermore, the forger can never win this final game, as any valid forgery will be rejected by our abort condition. This is captured in the following two lemmas.

Lemma 4. *There exists a distinguisher $\mathcal{D}_2$ against the $\Phi\mathsf{HA}[\lambda]$, which runs in time $t = t + 2^8 \cdot q_h \cdot t_{\mathsf{RSA}}$ and such that $|\Pr[\mathsf{G}_3^{\mathcal{F}} \Rightarrow 1] - \Pr[\mathsf{G}_4^{\mathcal{F}} \Rightarrow 1]| = \mathbf{Adv}_{\mathcal{D}_2}^{\Phi\mathsf{HA}[\lambda]}$.*

Proof. From G_3 to G_4, we change from a lossy key $(N, \hat{e})$ to a normal key (N, e), however the oracles are identical in both games. We now build a distinguisher $\mathcal{D}_2$ against $\Phi\mathsf{HA}[\lambda]$, using these two games. The distinguisher runs $\mathcal{F}$ and simulates the three oracles $\mathtt{H}, \mathtt{G}, \mathsf{Sign}$ as described in games G_3 and G_4. Note that $\mathcal{D}_2$ does not require the signing key to simulate the oracles. After $\mathcal{F}$ calls **Finalize**, $\mathcal{D}_2$ returns the output of **Finalize**. Thus we can see that $\Pr[1 \leftarrow \mathcal{D}_2(N, e)] = \Pr\left[\mathsf{G}_4^{\mathcal{F}} \Rightarrow 1\right]$ and $\Pr\left[1 \leftarrow \mathcal{D}_2(N, \hat{e})\right] = \Pr\left[\mathsf{G}_3^{\mathcal{F}} \Rightarrow 1\right]$ Hence we have $\left|\Pr\left[\mathsf{G}_4^{\mathcal{F}} \Rightarrow 1\right] - \Pr\left[\mathsf{G}_3^{\mathcal{F}} \Rightarrow 1\right]\right| = |\Pr[1 \leftarrow \mathcal{D}_2(N, e)] - \Pr[1 \leftarrow \mathcal{D}_2(N, \hat{e})]| = \mathbf{Adv}_{\mathcal{D}_2}^{\Phi\mathsf{HA}[\lambda]}$. □

Lemma 5. $\Pr[\mathsf{G}_4 \Rightarrow 1] = 0$.

Proof. In G_4 we are now using an injective instance, i.e. $x^e \bmod N$ is a permutation. Since our signing function is now injective, any forgery implies a collision. This now implies that for any message-randomness pair, the simulated signature $\sigma_{(m,r)}$ is the only valid signature. Therefore whenever the forger is able to produce a valid forgery, the game outputs 0 due to it failing the check, hence
$\Pr[\mathsf{G}_4 \Rightarrow 1] = 0$. □

We now combine the results of Lemmas 1-5 and we get our final result

$$\Pr[\mathsf{G}_0^{\mathcal{F}} \Rightarrow 1] = \mathbf{Adv}_{\mathcal{D}_1}^{\Phi\mathsf{HA}[\lambda]} + \left(\frac{\eta}{\eta - 1}\right) \mathbf{Adv}_{\mathcal{D}_2}^{\Phi\mathsf{HA}[\lambda]} + \frac{(q_s + q_h)^2}{2^{\ell_{\mathtt{H}}}}$$

Since $\mathcal{D}_1$ and $\mathcal{D}_2$ run in the same time, they have at most advantage of ε', by assumption. This now gives us

$$\begin{aligned} \Pr[\mathsf{G}_0^{\mathcal{F}} \Rightarrow 1] &\leqslant \varepsilon' + \left(\frac{\eta}{\eta - 1}\right) \varepsilon' + \frac{(q_s + q_h)^2}{2^{\ell_{\mathtt{H}}}} \\ &= \left(\frac{2\eta - 1}{\eta - 1}\right) \cdot \varepsilon' + \frac{(q_h + q_s)^2}{2^{\ell_{\mathtt{H}}}}. \end{aligned}$$

The running comes from the expected running time required to sample a simulated signature. This completes the proof. □

Remark 1. Although at first reading it seems like the security of PSS00 matches that of PSS96, this is not true. The increased running time means that PSS00 loses and additional 8 bits of security, which need to be compensated for. This security loss comes from the work factor of our reductions. The work factor ($\mathcal{WF}$) of an adversary $\mathcal{A}$ is defined as $\mathcal{WF} = \frac{\varepsilon_{\mathcal{A}}}{t_{\mathcal{A}}}$. When we say we have κ-bit security, this translates to having a work factor of at least 2^{κ}. Since the reduction for PSS96 and PSS00 have the same advantage, but the time differs by 2^8, we have a loss of 8 bits of security. Suppose for some parameters we have a work factor of 2^{κ}, i.e. κ bits of security, for PSS96, then we have:

$$\frac{\varepsilon_{\mathtt{PSS00}}}{t_{\mathtt{PSS00}}} = \frac{\varepsilon_{\mathtt{PSS96}}}{2^8 \cdot t_{\mathtt{PSS96}}} = \frac{1}{2^8} \cdot \frac{\varepsilon_{\mathtt{PSS96}}}{t_{\mathtt{PSS96}}} \frac{2^{\kappa}}{2^8} = 2^{\kappa - 8}.$$

While it is important to note this loss, it is also to be aware of the context and the effect it has on currently deployed systems. It is indeed true that they are not as secure as we thought, but this loss in security is in most cases not very severe, in fact most schemes still remain secure despite this.

4.2 Mask Generation Functions

A common practice, and indeed the recommendation in the PKCS#1 Standard [RFC 8017], is to not have a separate hash function G with a large output size, but rather to take our hash function H and use a so-called*Mask generation Function* (MGF). The MGF MGF1 was introduced in PKCS#1 Version 2.0 [RFC 2437, Sec. 10.2.], and is still in use in Version 2.2 [RFC 8017, App B.2.1]. MGF1 work by repeatedly hashing the same input with a 32-bit counter repeatedly until the total output size is reached. Consider a hash function H with output length $\ell_{\mathtt{H}}$; to extend this to an ℓ-bit hash function, we evaluate H $\Lambda = \left\lceil \frac{\ell}{\ell_{\mathtt{H}}} \right\rceil$ many times and take the ℓ most significant bits. we denote this as:

$$\mathtt{MGF1}[\mathtt{H}, \ell](x) = \mathtt{MSBs}\left(\mathtt{H}(x||\langle 0\rangle_{32})||\mathtt{H}(x||\langle 1\rangle_{32})|| \ldots ||\mathtt{H}(x||\langle \Lambda_{\mathtt{G}}\rangle_{32}), \ell_{\mathtt{G}}\right).$$

What this means of the case of PSS00 is that we have to evaluate H $\Lambda_{\mathtt{G}} = \left\lceil \frac{\ell_{\mathtt{G}}}{\ell_{\mathtt{H}}} \right\rceil$ may times. We then take the $\ell_{\mathtt{G}}$ most significant bits of the output, i.e.

$$\mathtt{G}(x) = \mathtt{MGF1}[\mathtt{H}, \ell_{\mathtt{G}}](x) = \mathtt{MSBs}\left(\mathtt{H}(x||\langle 0\rangle_{32})||\mathtt{H}(x||\langle 1\rangle_{32})|| \ldots ||\mathtt{H}(x||\langle \Lambda\rangle_{32}), \ell\right).$$

This complicates the proof slightly as we now have to program the same oracle multiple times for each query, instead of programming two oracles once. Additionally, we need to make case distinctions as to if a hash call is the "inner" hash, the "outer" hash or an MGF1 call and program the oracle appropriately. We make the assumption that that the first octect of our message is non-zer0. We now present the proof of PSS00 with one hash function. We believe this condition is quite reasonable, as it quite natural to use the minimal encoding of any message, i.e. without any superfluous leading 0 bits.

4.3 Proof with 1 Hash function

Theorem 2. *Assume* RSA *is a regular* (η, t', ε')*-lossy trapdoor permutation for* $\eta \geq 2$. *Then, for any* $(q_h, q_s, \ell_{\mathtt{R}}, \ell_{\mathtt{H}}), \mathtt{PSS0}[2, \lambda, \ell_{\mathtt{R}}, \ell_{\mathtt{H}}]$ *is* $(t, \varepsilon, q_h, q_s)$-UF-CMA *secure in the Random Oracle Model, where*

$$\begin{aligned} \varepsilon &\leqslant \left(\frac{2\eta - 1}{\eta - 1}\right) \cdot \varepsilon' + \frac{(q_h + q_s)^2}{2^{\ell_{\mathtt{H}}}} \\ t' &\approx t + 2^8 \cdot (q_h + q_s) \cdot t_{\mathsf{RSA}}, \end{aligned}$$

and t_{RSA} *is the time to compute one modular exponentiation.*

Proof. We prove our games through a series of game hops, which we describe in Figures 5a and 5b. We move from our standard EUF-CMA-ROM game to a game where the adversary has an advantage of 0. We show that each transition from game to game is negligible via Lemmas 6-10

The first change we make is to move to Game $\overline{\mathsf{G}}_1$, where we no longer generate the public key and signing key, but we use an *injective* $\Phi\mathsf{HA}[\lambda]$ instance as our public key and we program our random oracles to simulate signatures. This works, unless we get a collision in one of our hash functions, which happens with a small probability. This is captured by the following lemma.

Lemma 6. $\Pr[\overline{\mathsf{G}}_1^{\mathcal{F}} \Rightarrow 1] = \Pr[\overline{\mathsf{G}}_0^{\mathcal{F}} \Rightarrow 1] - \frac{(q_h+q_s)^2}{2^{\ell_{\mathtt{H}}}}$.

Proof. In $\overline{\mathsf{G}}_1$ we modify the `H` oracle and the signing oracle. First we distinguish between "signable" queries and "non-signable" queries. Only queries of the form $0^{64}||\mathtt{H}(m)||r \in \{0,1\}^{64+\ell_{\mathtt{H}}+\ell_{\mathtt{R}}}$ will be signed. Thus on these queries, the `H` oracle now works by sampling a random element $\sigma_{(m,r)} \in \mathbb{Z}_N^*$ and raises computes a candidate message representative $y = \sigma_{(m,r)}^e \bmod N$. This process is repeated until the trailing 16 bits of y are equal to 10111100[4] This takes an expected 2^8 resamplings. Once a valid message representative y is found, it is parsed as $0||\gamma_{(m,r)}||r^*_{(m,r)}||\omega_{(m,r)}||10111100$. We now check if $\omega_{(m,r)}$ has been previously queried to the `H` oracle, as an `MGF1` query. If it has, then we abort, as this would lead to an inconsistent oracle state. If this is not the case, we program the `H` oracle for each `MGF1` block, including padding the last block with a random value. The `H` oracle the stores the response $\omega_{(m,r)}$ and the signature $\sigma_{(m,r)}$. The signing oracle makes the requisite oracle calls, to ensure that all values are defined and then simply retrieves the signature and returns $\sigma_{(m,r)}$. It is clear to see that this indeed a valid signature. Since we are using the injective variant, the RSA function is a permutation, hence the response of `H` oracle is also correctly distributed. Hence, the simulation of signatures is correct, unless the reduction aborts. The probability of a collision is at most $\frac{(q_s+q_h)^2}{2^{\ell_{\mathtt{H}}}}$, since the adversary makes at most $q_h + q_s$ `MGF1` queries, implicit or explicit, to the `H`-oracle, as when we program one `MGF1` block, we program them all. This means we have at most $(q_h + q_s + 1)^2$ possible collisions from a total of $2^{\ell_{\mathtt{H}}}$ possible choices. Furthermore, we do not abort if collision matches the value already set, which happens with a probability $1 - \frac{1}{2^{\lambda-\ell_{\mathtt{H}}}}$. Hence the total probability that we abort is $\left(1 - \frac{1}{2^{\lambda-\ell_{\mathtt{H}}}}\right)\left(\frac{(q_h+q_s+1)^2}{2^{\ell_{\mathtt{H}}}}\right)$. Thus we have $\Pr[\overline{\mathsf{G}}_1^{\mathcal{F}} \Rightarrow 1] = \Pr[\overline{\mathsf{G}}_0^{\mathcal{F}} \Rightarrow 1] - \frac{(q_h+q_s+1)^2}{2^{\ell_{\mathtt{H}}}}$. □

From this point the proof proceeds as the proof of Theorem 1, thus we omit the proofs for Lemmas 7-10, as the proofs are close to that of Lemmas 2-5. As such, we find such a repetition non-instructive and we simply describe the game transitions and state the Lemmas.

The next change to game $\overline{\mathsf{G}}_2$, where we switch from an *injective* $\Phi\mathsf{HA}[\lambda]$ instance to a *lossy* $\Phi\mathsf{HA}[\lambda]$ instance. This change should not be noticed by the forger. If the forger does notice this change, then we can then use them to construct a distinguisher against the $\Phi\mathsf{HA}[\lambda]$.

Lemma 7. *There exists a distinguisher* $\overline{\mathcal{D}}_1$ *against the* $\Phi\mathsf{HA}[\lambda]$*, which runs in time* $t' = t + 2^8 \cdot q_h \cdot t_{\mathsf{RSA}}$ *and such that* $\left|\Pr[\overline{\mathsf{G}}_1^{\mathcal{F}} \Rightarrow 1] - \Pr[\overline{\mathsf{G}}_2^{\mathcal{F}} \Rightarrow 1]\right| = \mathbf{Adv}^{\Phi\mathsf{HA}[\lambda]}_{\overline{\mathcal{D}}_1}$.

[4] 10111100 is the binary for `0xbc` as specified in the standard [IEEE 1363].

Game G_0

Initialize(1^λ)
$(\mathsf{pk}, \mathsf{sk}) \leftarrow_\$ \mathsf{PSS00}[2, \lambda, \ell_\mathsf{R}, \ell_\mathsf{H}].\mathsf{KeyGen}(1^\lambda)$
return pk

Oracle H(x)
if $(x, \cdot) \in \mathcal{H}$
  fetch $(x, \omega_x) \in \mathcal{H}$
  return ω_x
else
  $\omega_x \in_R \{0,1\}^{\ell_\mathsf{H}}$
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{(x, \omega_x)\}$
  return ω_x

Sign(m)
$\mathcal{M} \leftarrow \mathcal{M} \cup \{m\}$
$\sigma \leftarrow_\$ \mathsf{PSS00}[2, \ell_\mathsf{R}, \ell_\mathsf{H}].\mathsf{Sign}(\mathsf{sk}, m)$
return σ

Finalize(m^*, σ^*)
if $(\mathsf{PSS00}[2, \ell_\mathsf{R}, \ell_\mathsf{H}].\mathsf{Verify}(\mathsf{pk}, m^*, \sigma^*) == 1) \wedge (m^* \notin \mathcal{M})$
  return 1
else
  return 0

(a) Game G_0 for Theorem 2

Games $\overline{\mathsf{G}}_1 - \overline{\mathsf{G}}_4$

Initialize(1^λ)
$(N, \mathfrak{e}_{inj}, \mathfrak{e}_{los}) \leftarrow_\$ \Phi\mathsf{HA}[\lambda]$
$e = \mathfrak{e}_{inj}$ //**Injective keys for $\overline{\mathsf{G}}_1, \overline{\mathsf{G}}_4$**
$e = \mathfrak{e}_{los}$ //**Lossy keys for $\overline{\mathsf{G}}_2, \overline{\mathsf{G}}_3$**
return $\mathsf{pk} = (N, \hat{e})$

Oracle H(x)
if $(\mathcal{H}[x]$ is defined)
  fetch $\mathcal{H}[x] = (\omega_x, \sigma_x)$
  return ω_x

else if $(|x| = \ell_\mathsf{H} + \ell_\mathsf{R} + 64$ and $\mathtt{MSBs}(x, 64) = 0^{64})$
//PSS00 **singing query**
  parse $x = 0^{64}||m||r$
  repeat
    $\sigma_{(m,r)} \in_R \mathbb{Z}_N^*$
    $y_{(m,r)} = \sigma^e \bmod \widehat{N}$
  until $\mathtt{LSBs}(y_{(m,r)}, 16) = 10111100$
  parse $y_{(m,r)} = 0||\gamma_{(m,r)}||r^*_{(m,r)}||\omega_{(m,r)}||10111100$
  if $(\mathcal{H}[\omega_{(m,r)}||\langle 0\rangle_{32}]$ is defined)
    return $\perp$ //MGF1 **collision abort**
  $\alpha_{(m,r)} = \gamma_{(m,r)}||(r^*_{(m,r)} \oplus 1||r)$
  parse $\alpha_{(m,r)} = \alpha_0||\ldots||\alpha_{\Lambda_\mathfrak{G}-1}||\beta_{\Lambda_\mathfrak{G}}$
  //$\alpha_i \in \{0,1\}^{\ell_\mathsf{H}}, \beta_{\Lambda_\mathfrak{G}} \in \{0,1\}^{\ell_\mathfrak{G} - \ell_\mathsf{H}(\Lambda_\mathfrak{G}-1)}$
  $\beta' \in_R \{0,1\}^{\ell_\mathsf{H}\cdot\Lambda_\mathfrak{G} - \ell_\mathfrak{G}}$
  $\alpha_{\Lambda_\mathfrak{G}} = \beta_{\Lambda_\mathfrak{G}}||\beta'$
  for$(i \in [\![0, \Lambda_\mathfrak{G}]\!])$
    set $\mathcal{H}[\omega_{(m,r)}||\langle i\rangle_{32}] = (\alpha_i, \sigma_{m,r})$
  set $\mathcal{H}[x] = (\omega_{(m,r)}, \perp, \sigma_{m,r})$
  return $\omega_{(m,r)}$

else if$(\mathtt{LSBs}(x, 32) \in \{\langle 0\rangle_{32}, \ldots \langle \Lambda_\mathsf{F}\rangle_{32}\})$ then
//MGF1 **query**
  parse $x = m||\langle\kappa\rangle_{32}$
  $z_{m_1} \in_R \{0,1\}^{\ell_\mathfrak{G}}$
  if $(\mathcal{H}[(m_1||\langle 0\rangle_{32}]$ is defined)
    return $\perp$
  parse $z_{m_1} = z_0||\ldots||z_{\Lambda_\mathfrak{G}-1}||\zeta_{\Lambda_\mathfrak{G}}$
  $\zeta' \in_R \{0,1\}^{\ell_\mathsf{H}\cdot\Lambda_\mathfrak{G} - \ell_\mathfrak{G}}$
  $z_{\Lambda_\mathfrak{G}} = \zeta_{\Lambda_\mathfrak{G}}||\zeta'$
  for$(i \in [\![0, \Lambda_\mathsf{F}]\!])$
    set $\mathcal{H}[m_1||\langle i\rangle_{32}] = (z_i, \perp)$
  return z_κ

else //**Non-critical query**
  $\omega_x \in_R \{0,1\}^{\ell_\mathsf{H}}$
  set $\mathcal{H}[x] = (\omega_x, \perp)$

Finalize(m^*, σ^*)
if $m^* \in \mathcal{M}$
  return 0
$y = (\sigma^*)^e \mod N$
if$(\mathtt{LSBs}(y, 16)! = 10111100)$
  return 0
parse $y = 0||\gamma||r^*||\omega||10111100$
fetch $\mathcal{G}[\omega^*] = (\alpha_{\omega^*})$
$1||r = r^* \oplus \mathtt{LSBs}(\alpha_{\omega^*}, \ell_\mathsf{R} + 1)$
fetch $\mathcal{H}[m^*] = (\omega_{m^*}, \sigma_{m^*})$
fetch $\mathcal{H}[0^{64}||\omega_{m^*}||r] = (\omega^*, \sigma_{(m,r)})$
if $(\sigma^* = \sigma_{(m,r)})$ then //**Abort rule in $\overline{\mathsf{G}}_3, \overline{\mathsf{G}}_4$**
  return 0 //**Abort rule in $\overline{\mathsf{G}}_3, \overline{\mathsf{G}}_4$**
if $(\omega^* == \omega \wedge \mathtt{MSBs}(\alpha_{\omega^*}, \ell_\mathfrak{G} - \ell_\mathsf{R} - 1) == \gamma)$
  return 1
else
  return 0

(b) Games $\overline{\mathsf{G}}_1$-$\overline{\mathsf{G}}_4$ for Theorem 2

Figure 5: Games $\overline{\mathsf{G}}_0 - \overline{\mathsf{G}}_4$ for Proof of PSS00 with 1 Random Oracle.

The next change to game $\overline{\mathsf{G}}_2$, where we switch from an *injective* $\Phi\mathsf{HA}[\lambda]$ instance to a *lossy* $\Phi\mathsf{HA}[\lambda]$ instance. This change should not be noticed by the forger. If the forger does notice this change, then we can then use them to construct a distinguisher against the $\Phi\mathsf{HA}[\lambda]$.

We now move to Game $\overline{\mathsf{G}}_3$, where we add an additional abort condition to Finalize procedure. Here we reject any forgery that is the same as our simulated signature. Since we now have lossy signatures, there are many valid signatures, thus the adversary may submit a forgery that is different from our simulated signature. We show this in the following lemma.

Lemma 8. $\Pr[\overline{\mathsf{G}}_3^{\mathcal{F}} \Rightarrow 1] = \left(\frac{\eta-1}{\eta}\right) \Pr[\overline{\mathsf{G}}_2^{\mathcal{F}} \Rightarrow 1]$.

The last step in the proof is to move to Game $\overline{\mathsf{G}}_4$. Here we switch back to an *injective* $\Phi\mathsf{HA}[\lambda]$ instance, which makes our RSA function once again a permutation. As with the change from $\overline{\mathsf{G}}_1$ to $\overline{\mathsf{G}}_2$, this is unnoticed by the forger, or we can use the forger to build a distinguisher. Furthermore, the forger can never win this final game, as any valid forgery will be rejected by our abort condition. This is captured in the following two lemmas.

Lemma 9. *There exists a distinguisher* $\overline{\mathcal{D}}_2$ *against the* $\Phi\mathsf{HA}[\lambda]$*, which runs in time* $t = t + 2^8 \cdot q_h \cdot t_{\mathsf{RSA}}$ *and such that* $|\Pr[\overline{\mathsf{G}}_3^{\mathcal{F}} \Rightarrow 1] - \Pr[\overline{\mathsf{G}}_4^{\mathcal{F}} \Rightarrow 1]| = \mathbf{Adv}^{\Phi\mathsf{HA}[\lambda]}_{\overline{\mathcal{D}}_2}$.

Lemma 10. $\Pr[\overline{\mathsf{G}}_4 \Rightarrow 1] = 0$.

We now combine the results of Lemmas 6-10 and we get our final result

$$\Pr[\overline{\mathsf{G}}_0^{\mathcal{F}} \Rightarrow 1] = \mathbf{Adv}^{\Phi\mathsf{HA}[\lambda]}_{\overline{\mathcal{D}}_1} + \left(\frac{\eta}{\eta-1}\right) \mathbf{Adv}^{\Phi\mathsf{HA}[\lambda]}_{\overline{\mathcal{D}}_2} + \frac{(q_s+q_h)^2}{2^{\ell_{\mathtt{H}}}}$$

Since $\overline{\mathcal{D}}_1$ and $\overline{\mathcal{D}}_2$ run in the same time, they have at most advantage of ε', by assumption. This now gives us

$$\begin{aligned} \Pr[\overline{\mathsf{G}}_0^{\mathcal{F}} \Rightarrow 1] &\leqslant \varepsilon' + \left(\frac{\eta}{\eta-1}\right)\varepsilon' + \frac{(q_s+q_h)^2}{2^{\ell_{\mathtt{H}}}} \\ &= \left(\frac{2\eta-1}{\eta-1}\right)\cdot\varepsilon' + \frac{(q_h+q_s)^2}{2^{\ell_{\mathtt{H}}}}. \end{aligned}$$

The running comes from the expected running time required to sample a simulated signature. This completes the proof. □

5 The PKCS#1 v1.5 Signature Scheme

The PKCS#1 v1.5 signature scheme (`PKCS1`) was first publicly defined in Version 1.5 of the RSA PKCS#1 Standard [RFC 2313]. It is used in countless important applications, such as X.509 Certificates [RFC 4055], Secure/Multipurpose Internet Mail Extensions (S/MIME) [RFC 3770], PGP [RFC 4880], IPSec [RFC 4359], all TLS versions up to 1.2 [RFC 2246, RFC 4346, RFC 5246], JSON Web Signature [RFC 7515], W3C's XML Signature [RFC 3275] and many more. However, as previously stated, due to the lack of a security proof, `PSS00` was recommended as an alternative.

Unlike `PSS96` or `PSS00`, `PKCS1` is deterministic and it requires only a single hash function, which we will for convenience call `F` and we denote its output size with $\ell_{\mathtt{F}}$. While it is standard to only use two prime factors, the standard allows for an arbitrary number of prime factors. The only know security proof due to Jager et al. [JKM18a, JKM18b] only holds for `PKCS1` with *at least* three prime factors. As with `PSS96` and `PSS00`, we denote the number of prime factors as k and we incorporate these parameters into our signatures name, i.e. `PKCS1`$[k, \ell_{\mathtt{F}}]$. We now recall the `PKCS1` scheme as presented by Jager et al. [JKM18a, JKM18b] in Figure 6a.

For completeness, we briefly recall the theorem of Jager et al. [JKM18a, JKM18b], for a detailed proof and discussion of the results, we refer the reader to the original paper [JKM18a, JKM18b].

Scheme $\texttt{PKCS1}$ $[k, \ell_{\texttt{F}}]$

KeyGen(1^λ)
$N = 1, \varphi(N) = 1$
for $i \in [\![1, k]\!]$
  $p_i \in_R \mathbb{P}[\lambda/k]$
  $N = N \cdot p_i$
  $\varphi(N) = \varphi(N) \cdot (p_i - 1)$
end for
$e \in_R \mathbb{Z}_N^*, \gcd(e, \varphi(N)) = 1$
pick hash function $\texttt{F} : \mathbb{M} \to \{0,1\}^{\ell_{\texttt{F}}}$
Look up identifier $\texttt{ID}_{\texttt{F}}$ for $\texttt{F}$
$\texttt{PAD} = 0^{15}||1^{\lambda - \ell_{\texttt{F}} - |\texttt{ID}_{\texttt{F}}| - 23}||0^8||\texttt{ID}_{\texttt{F}}$
return $(\mathsf{pk} = (N, e, \texttt{PAD}, \texttt{F}), \mathsf{sk} = (p_1, \ldots, p_k))$

Sign(sk, m)
$z \leftarrow \texttt{F}(m)$
$y = \texttt{PAD}||z$
return $\sigma = y^{1/e} \mod N$

Verify(pk, m, σ)
$y' = \sigma^e \mod N$
$z' \leftarrow \texttt{H}(m)$
if $(\texttt{PAD}||z' == y')$
  return 1
else
  return 0

(a) RSA PKCS#1 v1.5

Scheme $\mathsf{RSA\text{-}TLS\text{-}SIB}[k, \ell_{\texttt{R}}, \ell_{\texttt{H}}, \ell_{\texttt{F}}]$

algorithm KeyGen(1^λ)
$N = 1, \varphi(N) = 1$
for $i \in [\![1, k]\!]$
  $p_i \in_R \mathbb{P}[\lambda/k]$
  $N = N \cdot p_i$
  $\varphi(N) = \varphi(N) \cdot (p_i - 1)$
end for
$e \in_R \mathbb{Z}_N^*, \gcd(e, \varphi(N)) = 1$
pick hash functions
  $\texttt{H} : \mathbb{M}_2 \to \{0,1\}^{\ell_{\texttt{H}}}$
  $\texttt{G} : \{0,1\}^{\ell_{\texttt{H}}} \to \{0,1\}^{\ell_{\texttt{G}} = \lambda - \ell_{\texttt{H}} - 9}$
  $\texttt{F} : \mathbb{M}_1 \to \{0,1\}^{\ell_{\texttt{F}}}$
Look up identifier $\texttt{ID}_{\texttt{F}}$ for $\texttt{F}$
$\texttt{PAD} = 0^{15}||1^{\lambda - \ell_{\texttt{F}} - |\texttt{ID}_{\texttt{F}}| - 23}||0^8||\texttt{ID}_{\texttt{F}}$
$pk_1 = (N, e, \texttt{PAD}, \texttt{F}), \mathsf{pk}_2 = (N, e, \texttt{G}, \texttt{H})$
$\mathsf{sk}_1 = \mathsf{sk}_2 = (p_1, \ldots, p_k)$
return $((\mathsf{pk}_1, \mathsf{sk}_1), (\mathsf{pk}_2, \mathsf{sk}_2))$

Sign$_1(\mathsf{sk}_1, m)$
$z \leftarrow \texttt{F}(m)$
$y = \texttt{PAD}||z$
return $\sigma = y^{1/e} \mod N$

Verify$_1(\mathsf{pk}_1, m, \sigma)$
$y = \sigma^e \mod N$
$z \leftarrow \texttt{F}(m)$
if $(\texttt{PAD}||z == y)$
  return 1
else
  return 0

algorithm Sign$_2(\mathsf{sk}_2, m)$
$\mu = \texttt{H}(m)$
$r \in_R \{0,1\}^{\ell_{\texttt{R}}}$
$\omega \leftarrow \texttt{H}(0^{64}||\mu||r)$
$r^* \leftarrow \texttt{G}_2(\omega) \oplus 1||r$
$y = \sigma^e \mod N$
parse $y = 0||\texttt{G}_1(\omega)||r^*||\omega||10111100$
return $\sigma = y^{1/e} \mod N$

algorithm Verify$_2(\mathsf{pk}_2, m, \sigma)$
$y = \sigma^e \mod N$
parse y as $0||\gamma||r^*||\omega||10111100$
$r = r^* \oplus \texttt{G}_2(\omega)$
if $(\texttt{H}(0^{64}||\texttt{H}(m)||r) == \omega \wedge \texttt{G}_1(\omega) == \gamma)$
  return 1
else
  return 0

(b) RSA-TLS-Sibling Signatures

Figure 6: The $\texttt{PKCS1}$ and $\mathsf{RSA\text{-}TLS\text{-}SIB}$ signature schemes.

Theorem 3. [JKM18a, JKM18b] *Assume that* $\Phi\mathsf{HA}[\lambda]$ *is* (t', ε')*-hard and defines a* η*-regular lossy function. Then for any* (q_f, q_s), $\texttt{PKCS1}[3, 2\lambda]$ *is* $(t, \varepsilon, q_f, q_s)$-$\mathsf{UF\text{-}CMA}$ *secure in the Random Oracle Model, where*

$$\begin{aligned} \varepsilon &\leqslant \frac{2\eta - 1}{\eta - 1} \cdot \varepsilon' \\ t' &\approx t + q_f \cdot t_{\mathsf{Enc}}. \end{aligned}$$

Where t_{Enc} *is the running time of the* `Encode` *algorithm.*

We also recall the Encode algorithm by Jager et al. in Figure 7, as we will use it in our proofs. The Encode algorithm allows us to simulate `PKCS1` signatures. For a detailed discussion of the algorithm and its properties, we refer the reader to the original paper [JKM18a, JKM18b].

```
Encode (N, e, y, ℓ_F, PAD, p̃)
ν = ⌈log_2 N⌉, ρ = ⌈log_2 R⌉
z := 2^ℓ_F, k := 0
while (z ≥ 2^ℓ_F) and (k < n · 2^{n−ℓ_F}):
   k := k + 1
   s ←$ Z_N
   z := y s^e − 2^ℓ_F · PAD mod N
ŷ := 2^ℓ_F · PAD + z
if z < 2^ℓ_F then
   return (ŷ, s, z)
 else
   return ⊥.
```

Figure 7: The `Encode` algorithm due to Jager et al. [JKM18a, JKM18b]

Recall that the running time of the `Encode` algorithm is $\mathcal{O}(2^{\lambda - \ell_{\mathtt{F}}} \cdot \lambda^3 \cdot \log e) = \mathcal{O}(\lambda^4)$. We denote this running time as t_{Enc}.

6 Security proof of RSA-TLS-Sibling Signatures

Now that we have a security proof for `PSS00` and we have recalled the `PKCS1`, we can turn our attention to the RSA-TLS-Sibling Signatures ($\mathsf{RSA\text{-}TLS\text{-}SIB}$)[5] We will consider several possible cases in order to not only cover the current practices, but also potential future practices. The first case we consider is the ideal case, where the schemes are implemented with three independent hash functions in Section 6.1. This covers both the theoretical perspective and potential future usage. Our second proof in Section 6.2 covers the case where both schemes are implemented using only one hash function. We discuss the relevance of each proof after the proof itself. Finally in Section 6.3 we provide proofs for the case when we use 2 hash functions. To the best of our knowledge, this is not a common practice and its inclusion is simply for the sake of completeness.

6.1 Proof with 3 Hash Functions

We first consider the ideal case where have 3 hash functions, which we will call $\mathtt{H}, \mathtt{G}, \mathtt{F}$. We will use $\mathtt{H}, \mathtt{G}$ for `PSS00` and `F` for `PKCS1`, which we present in Figure 6b. While this proof seems relatively straight forward, it is a novel combination of our proof from Section 4.1 and the proof of Jager et al. (cf. Theorem 3) [JKM18a, JKM18b]. While it is possible that a `PSS00` signature could trivially give a `PKCS1` forgery, or indeed vice-versa, this is not a serious issue for our proof. This follows from the fact that we do not actually embed our challenges in the signatures, but in the public keys. Furthermore, our final game is one where the adversary has a success probability of 0, thus negating any trivial forgeries. We now present our result.

[5] We drop the Gen algorithm, for compactness, as we have very few system parameters beyond our keys.

Theorem 4. *Assume* RSA *is a regular* (η, t', ε')*-lossy trapdoor permutation for* $\eta \geq 2$*. Then, for any* $(q_h, q_g, q_f, q_{s_1}, q_{s_2}, \ell_{\mathtt{R}}, \ell_{\mathtt{H}}, \ell_{\mathtt{F}})$, RSA-TLS-SIB$[3, \lambda, \ell_{\mathtt{R}}, \ell_{\mathtt{H}}]$ *is* $(t, \varepsilon, q_h, q_g, q_f, q_{s_1}, q_{s_2})$-SIB-UF *secure in the Random Oracle Model, where*

$$\begin{aligned} \varepsilon &\leqslant \left(\frac{2\eta - 1}{\eta - 1}\right) \cdot \varepsilon' + \frac{(q_h + q_s)^2}{2^{\ell_{\mathtt{H}}}} \\ t' &\approx t + 2^8 \cdot (q_h + q_{s_1}) \cdot t_{\mathsf{RSA}} + (q_f + q_{s_2}) \cdot t_{\mathsf{Enc}}, \end{aligned}$$

and t_{RSA} *is the time to compute one modular exponentiation and* t_{Enc} *is the time to compute the Encode algorithm.*

Proof. We prove our theorem through a series of game hops. These game hops are captured in Lemmas 11-16. For compactness, we will only show the proof of Lemmas 11 & 12, as the proofs for Lemmas 13-15 are very close to that of Lemmas 2-5, thus including them here would not be instructive.

The fist change we make is that we now program our random oracles so that we are able to simulate Sign_2 (PSS00) signatures without needing the signing key. This is similar to how we simulate signatures in Lemma 1, with the additional issue of making sure that Sign_1 queries are unchanged.

Lemma 11. $\Pr[\widehat{\mathsf{G}}_1^{\mathcal{F}} \Rightarrow 1] = \Pr[\widehat{\mathsf{G}}_0^{\mathcal{F}} \Rightarrow 1] - \frac{(q_h+q_s)^2}{2^{\ell_{\mathtt{H}}}}$.

Proof. In $\widehat{\mathsf{G}}_1$ we modify the H oracle and the Sign_2 oracle such that Sign_2 Oracle no longer needs the signing key. First we distinguish between "signable" queries and "non-signable" queries. Only queries of the form $0^{64}||\mathtt{H}(m)||r \in \{0,1\}^{64+\ell_{\mathtt{H}}+\ell_{\mathtt{R}}}$ will be signed. Thus on these queries, the H oracle now works by sampling a random element $\sigma_{(m,r)} \in \mathbb{Z}_N^*$ and raises computes a candidate message representative $y = \sigma_{(m,r)}^e \bmod N$. This process is repeated until the trailing 16 bits of y are equal to 10111100[6] This takes an expected 2^8 resamplings. Once a valid message representative y is found, it is parsed as $0||\gamma_{(m,r)}||r^*_{(m,r)}||\omega_{(m,r)}||10111100$. We now check if $\omega_{(m,r)}$ has been previously queried to the G oracle. If it has, then we abort, as this would lead to an inconsistent oracle state. If this is not the case, we program the G oracles to return this value. The H oracle the stores the response $\omega_{(m,r)}$ and the signature $\sigma_{(m,r)}$. The signing oracle makes the requisite oracle calls, to ensure that all values are defined and then simply retrieves the signature and returns $\sigma_{(m,r)}$. It is clear to see that this indeed a valid signature. Since we are using the injective variant, the RSA function is a permutation, hence the response of H oracle is also correctly distributed. Hence, the simulation of signatures is correct, unless the reduction aborts. The probability of a collision is at most $\frac{(q_s+q_h)^2}{2^{k_1}}$, since the adversary makes at most $q_h + q_s$ queries, implicit or explicit, to the G-oracle, giving at most $(q_h + q_s + 1)^2$ possible collisions from a total of $2^{\ell_{\mathtt{H}}}$ possible choices. Furthermore, we do not abort if collision matches the value already set, which happens with a probability $1 - \frac{1}{2^{\lambda-\ell_{\mathtt{H}}}}$. Hence the total probability that we abort is $\left(1 - \frac{1}{2^{\lambda-\ell_{\mathtt{H}}}}\right)\left(\frac{(q_h+q_s+1)^2}{2^{\ell_{\mathtt{H}}}}\right)$. Thus we have $\Pr[\widehat{\mathsf{G}}_1^{\mathcal{F}} \Rightarrow 1] = \Pr[\widehat{\mathsf{G}}_0^{\mathcal{F}} \Rightarrow 1] - \frac{(q_h+q_s+1)^2}{2^{\ell_{\mathtt{H}}}}$. □

The next change we make is to program the random oracle so that we can simulate Sign_1 (PKCS1) queries without need the signing key. This is similar to the first step in the proof by Jager et al. [JKM18a, JKM18b].

Lemma 12. $\Pr[\widehat{\mathsf{G}}_2^{\mathcal{F}} \Rightarrow 1] = \Pr[\widehat{\mathsf{G}}_1^{\mathcal{F}} \Rightarrow 1]$.

Proof. We now modify the Initialize procedure, so that we no longer uses the RSA-TLS-Sibling Signature generation, but we get an injective ΦH instance. We bring this up a 3-prime modulus by sampling an appropriate prime number r and multiplying our modulus with that. Notices that the Sign_2 oracle does not need to be changed and still performs correctly. Now we need to take care of the Sign_1 oracle, which we do by modifying the F oracle. In $\widehat{\mathsf{G}}_2$ we modify the F oracle such that on any m, it now "precomputes" a signature for m using the Encode algorithm. It can be seen that all our signatures will verify due to the fact that $\sigma_m^e = \mathtt{PAD}||z_m$ for all m. Thus our simulation of the signatures is correct. Since Encode gives us

[6] 10111100 is the binary for 0xbc as specified in the standard [IEEE 1363].

Game $\widehat{\mathsf{G}}_0$

procedure Initialise
$(\mathsf{pk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{RSA\text{-}TLS\text{-}SIB}[2, \ell_{\mathtt{R}}, \ell_{\mathtt{H}}, \ell_{\mathtt{F}}].\mathsf{KeyGen}(1^{\lambda})$
return pk
Oracle $\mathbf{H}(x)$
if ($\mathcal{H}[x]$ is defined)
    fetch $\mathcal{H}[x] = (\omega_x)$
    return ω_x
else
    $\omega_x \in_R \{0,1\}^{\ell_{\mathtt{H}}}$
    set $\mathcal{H}[x] = (\omega_x)$

Oracle $\mathbf{G}(x)$
if ($\mathcal{G}[x]$ is defined)
    fetch $\mathcal{G}[x] = (\alpha_x)$
    return α_x
else
    $\alpha_x \in_R \{0,1\}^{\ell_{\mathtt{H}}}$
    set $\mathcal{H}[x] = (\alpha_x)$

Oracle $\mathbf{F}(x)$
if ($\mathcal{F}[x]$ is defined)
    fetch $\mathcal{F}[x] = (z_x)$
    return z_x
else
    $z_x \in_R \{0,1\}^{\ell_{\mathtt{F}}}$
    set $\mathcal{F}[x] = (z_x)$

Sign1(m)
$\mathcal{M}_1 \leftarrow \mathcal{M}_1 \cup \{m\}$
$\sigma \leftarrow \mathsf{RSA\text{-}TLS\text{-}SIB.Sign}_1(\mathsf{sk}, m)$
return σ

Sign2(m)
$\mathcal{M}_2 \leftarrow \mathcal{M}_2 \cup \{m\}$
$\sigma \leftarrow \mathsf{RSA\text{-}TLS\text{-}SIB.Sign}_2(\mathsf{sk}, m)$
return σ

Finalize(m^*, σ^*, i^*)
if($i^* = 1$) then
    if ($\mathsf{RSA\text{-}TLS\text{-}SIB.Verify}_1(\mathsf{pk}, m^*, i^*) == 1 \wedge m^* \notin \mathcal{M}_1$) then
        return 1
    else
        return 0
else if($i^* = 2$) then
    if ($\mathsf{RSA\text{-}TLS\text{-}SIB.Verify}_2(\mathsf{pk}, m^*, i^*) == 1 \wedge m^* \notin \mathcal{M}_2$) then
        return 1
    else
        return 0
else
    return 0

Figure 8: Game $\widehat{\mathsf{G}}_0$ for proof of RSA-TLS-SIB with 3 hash functions (Theorem 4)

Game $\widehat{\mathsf{G}}_1$

Initialise
$(\mathsf{pk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{RSA\text{-}TLS\text{-}SIB}[2, \ell_{\mathtt{R}}, \ell_{\mathtt{H}}, \ell_{\mathtt{F}}].\mathsf{KeyGen}(1^{\lambda})$
return pk

Oracle H(x)
if $(\mathcal{H}[x]$ is defined)
  fetch $\mathcal{H}[x] = (\omega_x, \sigma_x)$
  return ω_x
else if $(|x| = \ell_{\mathtt{H}} + \ell_{\mathtt{R}} + 64$ and $\mathtt{MSBs}(x, 64) = 0^{64})$
  parse $x = 0^{64}||m||r$
  repeat
    $\sigma_{(m,r)} \in_R \mathbb{Z}_N$
    $y_{(m,r)} = \sigma^e_{(m,r)} \bmod N$
  until $\mathtt{LSBs}(y_{(m,r)}, 16) = 10111100$
  Parse $y_{(m,r)} = 0||\gamma_{(m,r)}||r^*_{(m,r)}||\omega_{(m,r)}||10111100$
  if $(\mathcal{G}[\omega_{(m,r)}]$ is defined)
    return $\perp$
  $\alpha_{m,r} = \gamma_{m,r}||(r^*_{(m,r)} \oplus 1||r)$
  set $\mathcal{G}[\omega_{(m,r)}] = (\alpha_{(m,r)})$
  set $\mathcal{H}[x] = (\omega_{(m,r)}, \sigma_{(m,r)}, \perp)$
  return $\omega_{(m,r)}$
else
  $\omega_x \in_R \{0,1\}^{\ell_{\mathtt{H}}}$
  set $\mathcal{H}[x] = (\omega_x, \perp, \perp)$
  return ω_x

Sign1(m)
$\mathcal{M}_1 \leftarrow \mathcal{M}_1 \cup \{m\}$
$\sigma \leftarrow \mathsf{RSA\text{-}TLS\text{-}SIB.Sign}_1(\mathsf{sk}, m)$
return σ

Sign2(m)
$\mathcal{M}_2 \leftarrow \mathcal{M}_2 \cup \{m\}$
call $\mathtt{H}(m)$
fetch $\mathcal{H}[m] = (\omega_m)$
$r \in_R \{0,1\}^{\ell_{\mathtt{R}}}$
call $\mathtt{H}(0^{64}||\omega_m||r)$
fetch $\mathcal{H}[0^{64}||\omega_m||r] = (\omega, \sigma_1, \sigma_2)$
return σ_2

Oracle G(x)
if $(\mathcal{G}[x]$ is defined)
  fetch $\mathcal{G}[x] = (\alpha_x)$
  return α_x
else
  $\omega_x \in_R \{0,1\}^{\ell_{\mathtt{H}}}$
  set $\mathcal{H}[x] = (\omega_x)$

Oracle F(x)
if $(\mathcal{F}[x]$ is defined)
  fetch $\mathcal{F}[x] = (z_x)$
  return z_x
else
  $z_x \in_R \{0,1\}^{\ell_{\mathtt{F}}}$
  set $\mathcal{F}[x] = (z_x)$

Finalize(m^*, σ^*, i^*)
$y = (\sigma^*)^e \bmod N$
if$(i^* = 1)$ then
  if $(\mathsf{RSA\text{-}TLS\text{-}SIB.Verify}_1(\mathsf{pk}, m^*, i^*) == 1 \wedge m^* \notin \mathcal{M}_1)$ then
    return 1
  else
    return 0
else if $(i^* = 2)$ then
  if $m^* \in \mathcal{M}_2$
    return 0
  if$(\mathtt{LSBs}(y, 16)! = 10111100)$
    return 0
  fetch $\mathcal{H}[m^*] = (\omega_{m^*}, \perp, \perp)$
  parse $y = 0||\gamma||r^*||\omega^*||10111100$
  fetch $\mathcal{G}[\omega^*] = (\alpha_{\omega^*})$
  parse $1||r = r^* \oplus \mathtt{LSBs}(\alpha_{\omega^*}, \ell_{\mathtt{R}} + 1)$
  fetch $\mathcal{H}[0^{64}||\omega_{m^*}||r] = (\omega, \sigma_1, \sigma_2)$
  if $(\omega^* == \omega \wedge \mathtt{MSBs}(\alpha_{\omega^*}, \ell_{\mathtt{G}} - \ell_{\mathtt{R}} - 1) == \gamma)$
    return 1
  else
    return 0
else
  return 0

Figure 9: Game $\widehat{\mathsf{G}}_1$ for proof of RSA-TLS-SIB with 3 hash functions (Theorem 4)

Initialise
$(N, \mathfrak{e}_{inj}, \mathfrak{e}_{los}) \leftarrow_{\$} \Phi\mathsf{HA}[\lambda]$
$\tilde{p} \in_R \mathbb{P}[\lambda]; \gcd(\mathfrak{e}_{inj}, \tilde{p}-1) = \gcd(\mathfrak{e}_{los}, \tilde{p}-1) = 1$
$\widehat{N} = N \cdot \tilde{p}$
$e = \mathfrak{e}_{inj}$ //Injective keys for $\widehat{\mathsf{G}}_2, \widehat{\mathsf{G}}_5$
$e = \mathfrak{e}_{los}$ //Lossy keys for $\widehat{\mathsf{G}}_3, \widehat{\mathsf{G}}_4$
pick $\mathtt{ID_F}$
$\mathtt{PAD} = 0^{15}||1^{2\lambda-\ell-|\mathtt{ID_F}|-23}||0^8||\mathtt{ID_F}$
return $(\widehat{N}, e, \mathtt{PAD})$

Oracle H(x)
if $(\mathcal{H}[x]$ is defined)
  fetch $\mathcal{H}[x] = (\omega_x, \sigma_x)$
  return ω_x
else if $(|x| = \ell_{\mathsf{H}} + \ell_{\mathsf{R}} + 64$ and $\mathtt{MSBs}(x, 64) = 0^{64})$
  parse $x = 0^{64}||m||r$
  repeat
    $\sigma_{(m,r)} \in_R \mathbb{Z}_N$
    $y_{(m,r)} = \sigma^e_{(m,r)} \bmod N$
  until $\mathtt{LSBs}(y_{(m,r)}, 16) = 10111100$
  Parse $y_{(m,r)} = 0||\gamma_{(m,r)}||r^*_{(m,r)}||\omega_{(m,r)}||10111100$
  if $(\mathcal{G}[\omega_{(m,r)}]$ is defined)
    return $\perp$
  $\alpha_{m,r} = \gamma_{m,r}||(r^*_{(m,r)} \oplus 1||r)$
  set $\mathcal{G}[\omega_{(m,r)}] = (\alpha_{(m,r)})$
  set $\mathcal{H}[x] = (\omega_{(m,r)}, \sigma_{(m,r)})$
  return $\omega_{(m,r)}$
else
  $\omega_x \in_R \{0,1\}^{\ell_{\mathsf{H}}}$
  set $\mathcal{H}[x] = (\omega_x, \perp)$
  return ω_x

Sign1(m)
$\mathcal{M}_1 \leftarrow \mathcal{M}_1 \cup \{m\}$
call $\mathtt{F}(m)$
fetch $\mathcal{F}[m] = (z_m, \sigma_1)$
return σ_1

Sign2(m)
$\mathcal{M}_2 \leftarrow \mathcal{M}_2 \cup \{m\}$
call $\mathtt{H}(m)$
fetch $\mathcal{H}[m] = (\omega_m, \perp)$
$r \in_R \{0,1\}^{\ell_{\mathsf{R}}}$
call $\mathtt{H}(0^{64}||\omega_m||r)$
fetch $\mathcal{H}[0^{64}||\omega_m||r] = (\omega, \sigma_2)$
return σ_2

Oracle G(x)
if $(\mathcal{G}[x]$ is defined)
  fetch $\mathcal{G}[x] = (\alpha_x)$
  return α_x
else
  $\omega_x \in_R \{0,1\}^{\ell_{\mathsf{H}}}$
  set $\mathcal{H}[x] = (\omega_x)$

Oracle F(x)
if $(\mathcal{F}[m]$ is defined)
  fetch $\mathcal{F}[x] = (z_x)$
  return z_x
else
  $(\hat{y}, \sigma_x, z_x) \leftarrow_{\$} \mathtt{Encode}(N, e, 1, \mathtt{PAD}, \tilde{p})$
  set $\mathcal{F}[x] = (z_x, \sigma_x)$
  return z_x

Finalize(m^*, σ^*, i^*)
$y = (\sigma^*)^e \bmod N$
if $(i^* = 1)$ then
  if $m^* \in \mathcal{M}_1$
    return 0
  if$(\mathtt{MSBs}(y, |\mathtt{PAD}|) != \mathtt{PAD})$ then
    return 0
  fetch $\mathcal{F}(m^*) = (z_m, \sigma_1)$
  if $(\sigma^* = \sigma_1)$ then //Abort rule for $\widehat{\mathsf{G}}_4, \widehat{\mathsf{G}}_5$
    return 0 //Abort Rule for $\widehat{\mathsf{G}}_4, \widehat{\mathsf{G}}_5$
  parse $y = \mathtt{PAD}||z$
  if $(z = z_m)$ then
    return 1
  else
    return 0
else if $(i^* = 2)$ then
  if $m^* \in \mathcal{M}_2$
    return 0
  if$(\mathtt{LSBs}(y, 16) != 10111100)$
    return 0
  fetch $\mathcal{H}[m^*] = (\omega_{m^*}, \perp, \perp)$
  parse $y = 0||\gamma||r^*||\omega^*||10111100$
  fetch $\mathcal{G}[\omega^*] = (\alpha_{\omega^*})$
  parse $1||r = r^* \oplus \mathtt{LSBs}(\alpha_{\omega^*}, \ell_{\mathsf{R}} + 1)$
  fetch $\mathcal{H}[0^{64}||\omega_{m^*}||r] = (\omega, \sigma_2)$
  if $(\sigma^* = \sigma_2)$ then //Abort rule for $\widehat{\mathsf{G}}_4, \widehat{\mathsf{G}}_5$
    return 0 //Abort Rule for $\widehat{\mathsf{G}}_4, \widehat{\mathsf{G}}_5$
  if $(\omega^* == \omega \wedge \mathtt{MSBs}(\alpha_{\omega^*}, \ell_{\mathsf{G}} - \ell_{\mathsf{R}} - 1) == \gamma)$
    return 1
  else
    return 0
else
  return 0

Figure 10: Games $\widehat{\mathsf{G}}_2$-$\widehat{\mathsf{G}}_4$ for proof of RSA-TLS-SIB with 3 hash functions (Theorem 4)

uniformly distributed values s, z, the distribution of our hash queries in $\widehat{\mathsf{G}}_2$ is identical to the distribution in $\widehat{\mathsf{G}}_1$. Thus we have $\Pr[\widehat{\mathsf{G}}_2^{\mathcal{F}} \Rightarrow 1] = \Pr[\mathsf{G}_0^{\mathcal{F}} \Rightarrow 1]$. □

The next change to game $\widehat{\mathsf{G}}_3$, where we switch from an *injective* $\Phi\mathsf{HA}[\lambda]$ instance to a *lossy* $\Phi\mathsf{HA}[\lambda]$ instance. This change should not be noticed by the forger. If the forger does notice this change, then we can then use them to construct a distinguisher against the $\Phi\mathsf{HA}[\lambda]$.

Lemma 13. *There exists a distinguisher $\mathcal{D}_1$ against the $\Phi\mathsf{HA}[\lambda]$, which runs in time $t' = t + 2^8 \cdot (q_h + q_{s_1}) \cdot t_{\mathsf{RSA}} + (q_f + q_{s_2}) \cdot t_{\mathsf{Enc}}$ and such that $\left|\Pr[\widehat{\mathsf{G}}_2^{\mathcal{F}} \Rightarrow 1] - \Pr[\widehat{\mathsf{G}}_3^{\mathcal{F}} \Rightarrow 1]\right| = \mathbf{Adv}_{\mathcal{D}_1}^{\Phi\mathsf{HA}[\lambda]}$.*

We now move to Game $\widehat{\mathsf{G}}_4$, where we add an additional abort condition to Finalize procedure. Here we reject any forgery that is the same as our simulated signature. Since we now have lossy signatures, there are many valid signatures, thus the adversary may submit a forgery that is different from our simulated signature. We show this in the following lemma.

Lemma 14. $\Pr[\widehat{\mathsf{G}}_4^{\mathcal{F}} \Rightarrow 1] = \left(\frac{\eta-1}{\eta}\right) \Pr[\widehat{\mathsf{G}}_3^{\mathcal{F}} \Rightarrow 1]$.

Note that since the same key is used for both signature schemes, regardless of which scheme the adversary forges, he will still lose with a probability of $1/e$, which is the probability that the forger finds exactly the root the simulation chose.

The last step in the proof is to move to Game $\widehat{\mathsf{G}}_5$. Here we switch back to an *injective* $\Phi\mathsf{HA}[\lambda]$ instance, which makes our RSA function once again a permutation. As with the change from $\widehat{\mathsf{G}}_2$ to $\widehat{\mathsf{G}}_3$, this is unnoticed by the forger, or we can use the forger to build a distinguisher. Furthermore, the forger can never win this final game, as any valid forgery will be rejected by our abort condition. This is captured in the following two lemmas.

Lemma 15. *There exists a distinguisher $\mathcal{D}_2$ against the $\Phi\mathsf{HA}[\lambda]$, which runs in time $t = t + 2^8 \cdot (q_h + q_{s_1}) \cdot t_{\mathsf{RSA}} + (q_f + q_{s_2}) \cdot t_{\mathsf{Enc}}$ and such that $|\Pr[\widehat{\mathsf{G}}_4^{\mathcal{F}} \Rightarrow 1] - \Pr[\widehat{\mathsf{G}}_5^{\mathcal{F}} \Rightarrow 1]| = \mathbf{Adv}_{\mathcal{D}_2}^{\Phi\mathsf{HA}[\lambda]}$.*

Lemma 16. $\Pr[\widehat{\mathsf{G}}_5 \Rightarrow 1] = 0$.

We now combine the results of Lemmas 11-16 and we get our final result

$$\Pr[\widehat{\mathsf{G}}_0^{\mathcal{F}} \Rightarrow 1] = \mathbf{Adv}_{\mathcal{D}_1}^{\Phi\mathsf{HA}[\lambda]} + \left(\frac{\eta}{\eta-1}\right) \mathbf{Adv}_{\mathcal{D}_2}^{\Phi\mathsf{HA}[\lambda]} + \frac{(q_s+q_h)^2}{2^{\ell_{\mathrm{H}}}}$$

Since $\mathcal{D}_1$ and $\mathcal{D}_2$ run in the same time, they have at most advantage of ε', by assumption. This now gives us

$$\begin{aligned} \Pr[\widehat{\mathsf{G}}_0^{\mathcal{F}} \Rightarrow 1] &\leqslant \varepsilon' + \left(\frac{\eta}{\eta-1}\right)\varepsilon' + \frac{(q_s+q_h)^2}{2^{\ell_{\mathrm{H}}}} \\ &= \left(\frac{2\eta-1}{\eta-1}\right) \cdot \varepsilon' + \frac{(q_h+q_s)^2}{2^{\ell_{\mathrm{H}}}}. \end{aligned}$$

The running comes from the expected running time required to sample a simulated signature. This completes the proof.

□

The question remains if this is in any way representative of reality, to to which the answer is currently no, but there is hope. This proof requires the use of 3 prime moduli, which is not the standard. This is of course fixed by assuming that 2 prime moduli are indistinguishable from 3 prime moduli. All that remains is the issue of having three hash functions. The main argument for having a single hash function (and two Mask Generation Functions) is code complexity. This argument could be circumvented if the standards were expanded to allow the *customisable* SHA-3 [BDPA11] eXtendible Output Functions (XOFs) `cSHAKE128` and

cSHAKE256 [SP 800-185, FIPS 202]. In particular, one could simply implement one cSHAKE variant and then use the customisation string [SP 800-185, Sec 3.5] to get two, or even three, independent hash functions. This would be achieved by setting the hash functions, we use as below:

$$\begin{aligned}\mathtt{H}(x) &= \mathtt{cSHAKE}(x, \ell_{\mathtt{H}}, \text{“”}, \text{“Hash function H”})\\ \mathtt{G}(x) &= \mathtt{cSHAKE}(x, \ell_{\mathtt{G}}, \text{“”}, \text{“Hash function G”})\\ \mathtt{F}(x) &= \mathtt{cSHAKE}(x, \ell_{\mathtt{F}}, \text{“”}, \text{“Hash function F”})\end{aligned}$$

6.2 Proof with 1 Hash Function

We now move to the proof of the more complicated situation, that most closely mirrors reality. Firstly we assume that the eight most significant bits of any message are non-zero, or to put it another way, the first octet of our message is non-zero. This condition is quite reasonable, as it is more natural to use the minimal encoding of any message, but it also provides us with a good distinguishing criteria for our proofs. Furthermore, we consider the situation where we have only one hash function and this is reused to build all three has functions. This is achieved using the Mask Generation Function MGF1 from Section 4.2.

We now define $\mathtt{G}(x) = \mathtt{MGF1}[\mathtt{H}, \ell_{\mathtt{G}}](x)$, as in Section 4.3, and additionally, $\mathtt{F}(x) = \mathtt{MGF1}[\mathtt{H}, \ell_{\mathtt{F}}](x)$, which we plug into the construction in Figure 6b. This now gives us the closest scheme to reality that we can prove. The technical difficulty with this proof is that we now have one single random oracle instead of the multiple oracles previously. This leads to complications as where previously we were able to simply assign random values, we now need to take extra steps in our simulation to ensure internal consistency of the oracle. As with the proofs for PSS0 in Sections 4.1 and 4.3, we again attempt to partition our hash queries and deal with them in this manner. We will use $\Lambda_{\mathtt{G}} = \left\lceil \frac{\ell_{\mathtt{G}}}{\ell_{\mathtt{H}}} \right\rceil - 1$ and $\Lambda_{\mathtt{F}} = \left\lceil \frac{\ell_{\mathtt{F}}}{\ell_{\mathtt{H}}} \right\rceil - 1$, and assume wlog that $\Lambda_{\mathtt{G}} > \Lambda_{\mathtt{F}}$. This means that $\mathtt{F}(x) = \mathtt{MSBs}(\mathtt{G}(x), \ell_{\mathtt{F}})$, thus when we program one oracle we will have to implicitly program the other oracle. Additionally this means that any query to our H oracle can produce a “collision” in the functions wherein we wish to assign a value that has already been programmed.

Another issue to consider is what happens when we have a collision in message representatives, that is to say we have a message μ, where for some $j \in [\![1, \Lambda_{\mathtt{G}}]\!], m \in \{0,1\}^{\ell_{\mathtt{H}}}$, $r \in \{0,1\}^{\ell_{\mathtt{R}}}$, we have $\mu || \langle j \rangle_{32} = 0^{64} || m || r \in \{0,1\}^{\ell_{\mathtt{H}} + \ell_{\mathtt{R}} + 64}$. This would lead to some issues in the programming of the H-oracle, not least of which is that we need to fix $\Lambda_{\mathtt{F}}$ many signatures in advance to answer a single hash query. However, since we assume that all message start with a non-zero octet, this case can be ruled out. This case becomes an issue when we wish to use PSS0 with message recovery, but that discussion is out of scope for this article.

Theorem 5. *Assume* RSA *is a regular* (η, t', ε')*-lossy trapdoor permutation for* $\eta \geq 2$. *Then, for any* $(q_h, q_{s_1}, q_{s_2}, \ell_{\mathtt{R}}, \ell_{\mathtt{H}}, \ell_{\mathtt{F}})$, RSA-TLS-SIB$[3, \lambda, \ell_{\mathtt{R}}, \ell_{\mathtt{H}}, \ell_{\mathtt{F}}]$ *is* $(t, \varepsilon, q_h, q_{s_1}, q_{s_2})$-SIB-UF *secure in the Random Oracle Model, where*

$$\begin{aligned}\varepsilon &\leqslant \left(\frac{2\eta - 1}{\eta - 1}\right) \cdot \varepsilon' + \frac{(q_h + q_{s_1} + q_{s_2})^2}{2^{\ell_{\mathtt{H}}}}\\ t' &\approx t + (\mathcal{O}(q_h) + q_{s_1}) \cdot t_{\mathsf{RSA}} + (\mathcal{O}(q_h) + q_{s_2}) \cdot t_{\mathsf{Enc}},\end{aligned}$$

and t_{RSA} *is the time to compute a modular exponentiation and* t_{Enc} *is the time to run the* Encode *algorithm.*

Proof. We prove our theorem through a series of game hops. These game hops are captured in Lemmas 17-21. For compactness, we will only show the proof of Lemma 17, as the proofs for Lemmas 18-21 are very close to that of Lemmas 2-5, thus including them here would not be instructive.

The first change we make is that we now program our random oracles so that we are able to simulate Sign_2 (PSS0) and Sign_1 (PKCS1) signatures without needing the signing key. This is a careful combination to how we simulate signatures in Lemma 6 and the first step in the proof by Jager et al. [JKM18a, JKM18b].

Game $\widetilde{\mathsf{G}}_0$

procedure Initialise
$(\mathsf{pk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{RSA\text{-}TLS\text{-}SIB}[2, \ell_{\mathtt{R}}, \ell_{\mathtt{H}}, \ell_{\mathtt{F}}].\mathsf{KeyGen}(1^{\lambda})$
return pk

Oracle H(x)
if ($\mathcal{H}[x]$ is defined)
    fetch $\mathcal{H}[x] = (\omega_x)$
    return ω_x
else
    $\omega_x \in_R \{0,1\}^{\ell_{\mathtt{H}}}$
    set $\mathcal{H}[x] = (\omega_x)$

Sign1(m)
$\mathcal{M}_1 \leftarrow \mathcal{M}_1 \cup \{m\}$
$\sigma \leftarrow \mathsf{RSA\text{-}TLS\text{-}SIB.Sign}_1(\mathsf{sk}, m)$
return σ

Sign2(m)
$\mathcal{M}_2 \leftarrow \mathcal{M}_2 \cup \{m\}$
$\sigma \leftarrow \mathsf{RSA\text{-}TLS\text{-}SIB.Sign}_2(\mathsf{sk}, m)$
return σ

Finalize(m^*, σ^*, i^*)
if$(i^* = 1)$ then
    if ($\mathsf{RSA\text{-}TLS\text{-}SIB.Verify}_1(\mathsf{pk}, m^*, i^*) == 1 \wedge m^* \notin \mathcal{M}_1$) then
        return 1
    else
        return 0
else if$(i^* = 2)$ then
    if ($\mathsf{RSA\text{-}TLS\text{-}SIB.Verify}_2(\mathsf{pk}, m^*, i^*) == 1 \wedge m^* \notin \mathcal{M}_2$) then
        return 1
    else
        return 0
else
    return 0

Figure 11: Game $\widetilde{\mathsf{G}}_0$ for proof of RSA-TLS-SIB with 1 hash function (Theorem 5)

Sign1(m)
$\mathcal{M}_1 \leftarrow \mathcal{M}_1 \cup \{m\}$
call $\mathtt{H}(m||\langle 0\rangle_{32})$
fetch $\mathcal{H}[m||\langle 0\rangle_{32}] = (\omega_{m||\langle 0\rangle_{32}}, \sigma_1, \sigma_2)$
return σ_1

Sign2(m)
$\mathcal{M}_2 \leftarrow \mathcal{M}_2 \cup \{m\}$
call $\mathtt{H}(m)$
fetch $\mathcal{H}[m] = (\omega_m, \sigma_m)$
$r \in_R \{0,1\}^{\ell_\mathsf{R}}$
call $\mathtt{H}(0^{64}||\omega_m||r)$
fetch $\mathcal{H}[0^{64}||\omega_m||r] = (\omega_{(m,r)}, \sigma_1, \sigma_2)$
return σ_2

Finalize(m^*, σ^*, i^*)
$y = (\sigma^*)^e \mod N$
if$(i^* = 1)$ then
  if $(m^* \in \mathcal{M}_1)$
    return 0
  if$(\mathtt{MSBs}(y, |\mathtt{PAD}|) != \mathtt{PAD})$ then
    return 0
  parse $y = \mathtt{PAD}||z$
  fetch $\mathcal{H}(m^*||\langle 0\rangle_{32}) = (z_0, \sigma_1, \sigma_2)$
  $z_m = z_0$
  if $(\sigma^* = \sigma_1)$ then //**Abort rule for $\widetilde{\mathsf{G}}_3, \widetilde{\mathsf{G}}_4$**
    return 0 //**Abort Rule for $\widetilde{\mathsf{G}}_3, \widetilde{\mathsf{G}}_4$**
  for$(i \in [\![1, \Lambda_\mathsf{F}]\!])$
  fetch $\mathcal{H}(m||\langle 0\rangle_{32}) = (z_i, \sigma_1, \sigma_2)$
    $z_m = z_m||z_i$
  if $(z = \mathtt{MSBs}(z_m, \ell_\mathsf{F}))$ then
    return 1
  else
    return 0
else if $(i^* = 2)$ then
  if $(m^* \in \mathcal{M}_2)$
    return 0
  if$(\mathtt{LSBs}(y, 16) != 10111100)$
    return 0
  fetch $\mathcal{H}[m^*] = (\omega_{m^*}, \sigma_1, \sigma_2)$
  parse $y = 0||\gamma||r^*||\omega||10111100$
  for $(i \in [\![1, \Lambda_\mathsf{G}]\!])$
    fetch $\mathcal{H}[\omega^*||\langle i\rangle_{32}] = (\alpha_{\omega_i^*})$
    $\alpha_{\omega^*} = \alpha_{\omega^*}||\alpha_{\omega_i^*}$
  $1||r = r^* \oplus \mathtt{LSBs}(\alpha_{\omega^*}, \ell_\mathsf{R} + 1)$
  fetch $\mathcal{H}[0^{64}||\omega_{m^*}||r] = (\omega^*, \sigma_1, \sigma_2)$
  if $(\sigma^* = \sigma_2)$ then //**Abort rule for $\widetilde{\mathsf{G}}_3, \widetilde{\mathsf{G}}_4$**
    return 0 //**Abort Rule for $\widetilde{\mathsf{G}}_3, \widetilde{\mathsf{G}}_4$**
  if $(\omega^* == \omega \wedge \mathtt{MSBs}(\alpha_{\omega^*}, \ell_\mathsf{G} - \ell_\mathsf{R} - 1) == \gamma)$
    return 1
  else
    return 0
else
  return 0

procedure Initialise
$(N, \mathfrak{e}_{inj}, \mathfrak{e}_{los}) \leftarrow_\$ \Phi\mathsf{HA}[\lambda]$
$\tilde{p} \in_R \mathbb{P}[\lambda]; \gcd(\mathfrak{e}_{inj}, \tilde{p} - 1) = \gcd(\mathfrak{e}_{los}, \tilde{p} - 1) = 1$
$\widehat{N} = N \cdot \tilde{p}$
pick $\mathtt{ID}_\mathsf{F}$
$\mathtt{PAD} = 0^{15}||1^{2\lambda - \ell - |\mathtt{ID}_\mathsf{F}| - 23}||0^8||\mathtt{ID}_\mathsf{F}$
$e = \mathfrak{e}_{inj}$ //**Injective keys for $\widetilde{\mathsf{G}}_1, \widetilde{\mathsf{G}}_4$**
$e = \mathfrak{e}_{los}$ //**Lossy keys for $\widetilde{\mathsf{G}}_2, \widetilde{\mathsf{G}}_3$**
return $\mathsf{pk} = (\widehat{N}, e, \mathtt{PAD})$

Oracle H(x)
if $(\mathcal{H}[x]$ is defined)
  fetch $\mathcal{H}[x] = (\omega_x, \sigma_x)$
  return ω_x

else if $(|x| = \ell_\mathtt{H} + \ell_\mathsf{R} + 64$ and $\mathtt{MSBs}(x, 64) = 0^{64})$
//PSS00 **singing query**
  parse $x = 0^{64}||m_2||r$
  repeat
    $\sigma_{(m_2,r)} \in_R \mathbb{Z}_N^*$
    $y_{(m_2,r)} = \sigma^e \mod \widehat{N}$
  until $\mathtt{LSBs}(y_{(m_2,r)}, 16) = 10111100$
  parse $y_{(m_2,r)} = 0||\gamma_{(m_2,r)}||r^*_{(m_2,r)}||\omega_{(m_2,r)}||10111100$
  if $(\mathcal{H}[\omega_{(m_2,r)}||\langle 0\rangle_{32}]$ is defined)
    return $\perp$ //MGF1 **collision abort**
  $\alpha_{(m,r)} = \gamma_{(m_2,r)}||(r^*_{(m_2,r)} \oplus 1||r)$
  parse $\alpha_{(m_2,r)} = \alpha_0||\ldots||\alpha_{\Lambda_\mathtt{G}-1}||\beta_{\Lambda_\mathtt{G}}$
  //$\alpha_i \in \{0,1\}^{\ell_\mathtt{H}}, \beta_{\Lambda_\mathtt{G}} \in \{0,1\}^{\ell_\mathtt{G} - \ell_\mathtt{H}(\Lambda_\mathtt{G}-1)}$
  $\beta' \in_R \{0,1\}^{\ell_\mathtt{H}\cdot\Lambda_\mathtt{G} - \ell_\mathtt{G}}$
  $\alpha_{\Lambda_\mathtt{G}} = \beta_{\Lambda_\mathtt{G}}||\beta'$
  for$(i \in [\![0, \Lambda_\mathtt{G}]\!])$
    set $\mathcal{H}[\omega_{(m_2,r)}||\langle i\rangle_{32}] = (\alpha_i, \perp, \sigma_{m_2,r})$
  set $\mathcal{H}[x] = (\omega_{(m_2,r)}, \perp, \sigma_{m_2,r})$
  return $\omega_{(m_2,r)}$

else if$(\mathtt{LSBs}(x, 32) \in \{\langle 0\rangle_{32}, \ldots \langle \Lambda_\mathsf{F}\rangle_{32}\})$ then
//PKCS1 **signing query**
  parse $x = m_1||\langle\kappa\rangle_{32}$
  $(y_{m_1}, \sigma_{m_1}, z_{m_1}) \xleftarrow{\$} \mathtt{Encode}(N, e, 1, \mathtt{PAD}, \ell_\mathsf{F}, \tilde{p})$
  if $(\mathcal{H}[(m_1||\langle 0\rangle_{32}]$ is defined)
    return $\perp$
  parse $z_{m_1} = z_0||\ldots||z_{\Lambda_\mathsf{F}-1}||\zeta_{\Lambda_\mathtt{G}}$
  $\zeta' \in_R \{0,1\}^{\ell_\mathtt{H}\cdot\Lambda_\mathsf{F} - \ell_\mathsf{F}}$
  $z_{\Lambda_\mathtt{G}} = \zeta_{\Lambda_\mathtt{G}}||\zeta'$
  for$(i \in [\![0, \Lambda_\mathsf{F}]\!])$
    set $\mathcal{H}[m_1||\langle i\rangle_{32}] = (z_i, \sigma_{m_1}, \perp)$
  return z_κ

else //**Non-critical query**
  $\omega_x \in_R \{0,1\}^{\ell_\mathtt{H}}$
  set $\mathcal{H}[x] = (\omega_x, \perp, \perp)$

Figure 12: Games $\widehat{\mathsf{G}}_1$-$\widehat{\mathsf{G}}_4$ for prof of RSA-TLS-SIB with 1 hash function (Theorem 5)

Lemma 17. $\Pr[\widetilde{\mathsf{G}}_1^{\mathcal{F}} \Rightarrow 1] = \Pr[\widetilde{\mathsf{G}}_0^{\mathcal{F}} \Rightarrow 1] - \frac{(q_h+q_s)^2}{2^{\ell_{\mathtt{H}}}}$.

Proof. In $\widetilde{\mathsf{G}}_1$ we modify the `H` oracle and the signing oracle. First we distinguish between "signable" queries and "non-signable" queries, for both `PSS00` and `PKCS1`. Only queries of the form $0^{64}||m_2||r \in \{0,1\}^{64+\ell_{\mathtt{H}}+\ell_{\mathtt{R}}}$ will be signed by the `PSS00` oracle, Thus on these queries, the `H` oracle now works by sampling a random element $\sigma_{(m,r)} \in \mathbb{Z}_N^*$ and raises computes a candidate message representative $y = \sigma_{(m,r)}^e \bmod N$. This process is repeated until the trailing 16 bits of y are equal to 10111100[7] This takes an expected 2^8 resamplings. Once a valid message representative y is found, it is parsed as $0||\gamma_{(m,r)}||r^*_{(m,r)}||\omega_{(m,r)}||10111100$. We now check if $\omega_{(m,r)}||\langle 0\rangle_{32}$ has been previously queried to the `H` oracle, which would give us an inconsistency, in which case we abort. If this is not the case, we program the `H` oracle for all the values for $\omega_{(m,r)}||\langle 0\rangle_{32}, \ldots \omega_{(m,r)}||\langle \Lambda_{\mathtt{G}}\rangle_{32}$, taking care to pad the last entry to $\ell_{\mathtt{H}}$ bits long. Note that each entry will have two $\perp$ symbols to indicate these queries are non-signable for both schemes. Finally, the oracle stores $(\omega_{(m_2,r)}, \perp, \sigma_{(m_2,r)})$ and returns $\omega_{(m,r)}$.

If the query is not a `PSS00` signable query, but it is a `PKCS1` singable query, we run the Encode algorithm to get a signature σ_{m_1}. We then check if we have a collision in the `MGF1` with our `F` function query, in which case we abort. If not, then we take the hash value from our Encode algorithm and split it into the corresponding `MGF1` blocks, taking care to pad the final block. We store these hash values, along with the signature that we get from our Encode algorithm. Finally, we return the appropriate hash block.

Finally, if the query does not satisfy either criteria, then we consider it to be a non-signable query. The adversary will indeed make these queries to get the inner hash value of message. Since we do not have to simulate anything here, we can simply pick a random $\ell_{\mathtt{H}}$-bit string. We store this random value, a long with two $\perp$ symbols to indicate that this query is non-signable. We then return our randomly sampled value.

Now we can discuss the changes to the signing oracles. Both signing oracles make the requisite oracle calls, to ensure that all values are defined and then simply retrieves the signature and returns σ_1 for a `PKCS1` signature and σ_2 for a `PSS00` signature. It is clear to see that these are indeed a valid signatures, due the manner in which we computed our hash values. Since we are using the injective variant, the RSA function is a permutation, hence the response of `H` oracle is also correctly distributed. Hence, the simulation of signatures is correct, unless the reduction aborts. The probability of a collision is at most $\frac{(q_h+q_{s_1}+q_{s_2})^2}{2^{\ell_{\mathtt{H}}}}$, since the adversary makes at most $q_h + q_{s_1} + q_{s_2}$ queries, implicit or explicit, to the `H`-oracle, which could lead to a collision in the `MGF1` states for either `G` or `F`, giving at most $(q_h + q_{s_1} + q_{s_2})^2$ possible collisions from a total of $2^{\ell_{\mathtt{H}}}$ possible choices. Furthermore, we do not abort if collision matches the value already set, which happens with a probability $1 - \frac{1}{2^{\lambda-\ell_{\mathtt{H}}}}$. Hence the total probability that we abort is $\left(1 - \frac{1}{2^{\lambda-\ell_{\mathtt{H}}}}\right)\left(\frac{(q_h+q_{s_1}+q_{s_2})^2}{2^{\ell_{\mathtt{H}}}}\right)$. Thus we have $\Pr[\widetilde{\mathsf{G}}_1^{\mathcal{F}} \Rightarrow 1] = \Pr[\widetilde{\mathsf{G}}_0^{\mathcal{F}} \Rightarrow 1] - \frac{(q_h+q_{s_1}+q_{s_2})^2}{2^{\ell_{\mathtt{H}}}}$. □

The next change to game $\widetilde{\mathsf{G}}_2$, where we switch from an *injective* $\Phi\mathsf{HA}[\lambda]$ instance to a *lossy* $\Phi\mathsf{HA}[\lambda]$ instance. This change should not be noticed by the forger. If the forger does notice this change, then we can then use them to construct a distinguisher against the $\Phi\mathsf{HA}[\lambda]$.

Lemma 18. *There exists a distinguisher $\mathcal{D}_1$ against the $\Phi\mathsf{HA}[\lambda]$, which runs in time $t' = t + 2^8 \cdot q_h \cdot t_{RSA}$ and such that $\left|\Pr[\widetilde{\mathsf{G}}_1^{\mathcal{F}} \Rightarrow 1] - \Pr[\widetilde{\mathsf{G}}_2^{\mathcal{F}} \Rightarrow 1]\right| = \mathbf{Adv}_{\mathcal{D}_1}^{\Phi\mathsf{HA}[\lambda]}$.*

We now move to Game $\widetilde{\mathsf{G}}_3$, where we add an additional abort condition to Finalize procedure. Here we reject any forgery that is the same as our simulated signature. Since we now have lossy signatures, there are many valid signatures, thus the adversary may submit a forgery that is different from our simulated signature. We show this in the following lemma.

Lemma 19. $\Pr[\widetilde{\mathsf{G}}_3^{\mathcal{F}} \Rightarrow 1] = \left(\frac{\eta-1}{\eta}\right)\Pr[\widetilde{\mathsf{G}}_2^{\mathcal{F}} \Rightarrow 1]$.

[7] 10111100 is the binary for `0xbc` as specified in the standard [IEEE 1363].

The last step in the proof is to move to Game $\widetilde{\mathsf{G}}_4$. Here we switch back to an *injective* $\Phi\mathsf{HA}[\lambda]$ instance, which makes our RSA function once again a permutation. As with the change from $\widetilde{\mathsf{G}}_1$ to $\widetilde{\mathsf{G}}_2$, this is unnoticed by the forger, or we can use the forger to build a distinguisher. Furthermore, the forger can never win this final game, as any valid forgery will be rejected by our abort condition. This is captured in the following two lemmas.

Lemma 20. *There exists a distinguisher $\mathcal{D}_2$ against the $\Phi\mathsf{HA}[\lambda]$, which runs in time $t = t + 2^8 \cdot q_h \cdot t_{RSA}$ and such that $|\Pr[\widetilde{\mathsf{G}}_3^{\mathcal{F}} \Rightarrow 1] - \Pr[\widetilde{\mathsf{G}}_4^{\mathcal{F}} \Rightarrow 1]| = \mathbf{Adv}_{\mathcal{D}_2}^{\Phi\mathsf{HA}[\lambda]}$.*

Lemma 21. $\Pr[\widetilde{\mathsf{G}}_4 \Rightarrow 1] = 0$.

We now combine the results of Lemmas 17-21 and we get our final result

$$\Pr[\widetilde{\mathsf{G}}_0^{\mathcal{F}} \Rightarrow 1] = \mathbf{Adv}_{\mathcal{D}_1}^{\Phi\mathsf{HA}[\lambda]} + \left(\frac{\eta}{\eta - 1}\right) \mathbf{Adv}_{\mathcal{D}_2}^{\Phi\mathsf{HA}[\lambda]} + \frac{(q_s + q_h)^2}{2^{\ell_{\mathtt{H}}}}$$

Since $\mathcal{D}_1$ and $\mathcal{D}_2$ run in the same time, they have at most advantage of ε', by assumption. This now gives us

$$\begin{aligned} \Pr[\widetilde{\mathsf{G}}_0^{\mathcal{F}} \Rightarrow 1] &\leqslant \varepsilon' + \left(\frac{\eta}{\eta - 1}\right) \varepsilon' + \frac{(q_s + q_h)^2}{2^{\ell_{\mathtt{H}}}} \\ &= \left(\frac{2\eta - 1}{\eta - 1}\right) \cdot \varepsilon' + \frac{(q_h + q_s)^2}{2^{\ell_{\mathtt{H}}}}. \end{aligned}$$

What remains is to show the running time of our reduction. To simulate a `PSS0` signature, we need expected time $2^8 \cdot t_{\mathsf{RSA}}$ and to simulate a `PKCS1` signature we need expected time t_{Enc}. For each signature query, we need to make one hash call, which could potentially lead to a full signature simulation, for which we need time $q_{s_1} \cdot t_{\mathsf{Enc}} + q_{s_2} \cdot 2^8 \cdot t_{\mathsf{RSA}} = (q_{s_1} + q_{s_2}) \cdot t_{\mathsf{Enc}}$. To compute the time needed to simulate the hash queries is more complicated, as we cannot say in advance how may hash queries will lead to a `PSS0` simulation and how many will lead to a `PKCS1` simulation. However, since we know there are $\mathcal{O}(q_h)$ queries of each type, we can cover this with the term $\mathcal{O}(q_h)(t_{\mathsf{RSA}} + t_{\mathsf{Enc}}))$. Combining the two terms gives us a final running time of $(\mathcal{O}(q_h) + q_{s_1}) \cdot t_{\mathsf{RSA}} + (\mathcal{O}(q_h) + q_{s_2}) \cdot t_{\mathsf{Enc}}$. This completes the proof. □

6.3 Proofs with 2 Hash functions

For completeness, we now consider what one might call an intermediate case, where RSA is implemented with two hash function. This case is possibly the furthest from reality and has its own interesting complications. The first is that there are several possible combinations of which hash functions are used in which case. We will now give a brief overview, but omit the proofs, as once we describe how to program the Random Oracles, we simply apply the same technique as in the previous proofs. Repetition of the proof methodology would be non-instructive at this point.

Note that when we need to reduce the size of a hash function output, we take the most significant bits, so as to be consistent with `MGF1`. To the best of our knowledge, this would work equally well with the least significant bits.

Hash functions `H` and `F`

The first case is the simplest to deal with and is the case where we use one hash function per scheme. That is to say the we implement `PSS0` using only one hash function `H` and a Mask Generation Function (as in Section 6.2) and we implement `PKCS1` with another hash function `F`. This case can be shown to be secure by a simple combination of the proofs. We do not present the proof here as it would be non-instructive.

Theorem 6. *Assume that $\Phi\mathsf{HA}[\lambda]$ is (t', ε')-hard and defines a η-regular lossy function. Then for any $(q_h, q_g, q_{s_1}, q_{s_2})$, RSA-sibling is $(t, \varepsilon, q_h, q_g, q_{s_1}, q_{s_2})$-sibling secure in the Random Oracle Model, where*

$$\begin{aligned} \varepsilon &\leqslant \left(\frac{2\eta - 1}{\eta - 1}\right) \cdot \varepsilon' + \frac{(q_h + q_f + q_{s_2})^2}{2^{\ell_{\mathtt{H}}}} \\ t' &\approx t + (q_f + q_{s_1}) \cdot t_{Enc} + 2^8 \cdot (q_h + q_{s_2}) \cdot t_{RSA}. \end{aligned}$$

Hash functions H and G
The second case, which is more interesting is when we instantiate PSS00 with two hash functions $\mathtt{H}, \mathtt{G}$ and then we use G to instantiate PKCS1. Here we must take care of the disparity in the parameters as generally speaking $\ell_{\mathtt{G}} > \ell_{\mathtt{F}}$, thus we implicitly define a hash function $\mathtt{F}(x) = \mathtt{MSBs}(\mathtt{G}(x), \ell_{\mathtt{F}})$. In the proof however, we simply model the G random oracle. In this case, instead of uniformly sampling the values of G, the are chosen using the Encode function. This way we can produce a PKCS1 signature for anything queried to the G oracle. Once we have this, the proof then continues as before.

Theorem 7. Assume that $\Phi\mathsf{HA}[\lambda]$ *is* (t', ε')*-hard and defines a* η*-regular lossy function. Then for any* $(q_h, q_g, q_{s_1}, q_{s_2})$*, RSA-sibling is* $(t, \varepsilon, q_h, q_g, q_{s_1}, q_{s_2})$*-sibling secure in the Random Oracle Model, where*

$$\begin{aligned} \varepsilon &\leqslant \left(\frac{2\eta - 1}{\eta - 1}\right) \cdot \varepsilon' + \frac{(q_h + q_g + q_{s_2})^2}{2^{\ell_{\mathtt{H}}}} \\ t' &\approx t + (q_g + q_{s_1}) \cdot t_{\mathsf{Enc}} + (q_h + q_{s_2}) \cdot t_{\mathsf{RSA}}. \end{aligned}$$

Hash functions G and F
The last case, which is more interesting is when we instantiate PSS00 with two hash functions G and F and then we use F to instantiate PKCS1. Here we must take care of the disparity in the parameters as generally speaking $\ell_{\mathtt{F}} > \ell_{\mathtt{H}}$, thus we implicitly define a hash function $\mathtt{H}(x) = \mathtt{MSBs}(\mathtt{F}(x), \ell_{\mathtt{H}})$. Whenever a query is made to the F oracle, we do not randomly sample values, but use the Encode method and store the most significant bits. This allows us to also produce a PKCS1 signature for this message. Once we have this, the proof then continues as before.

Theorem 8. Assume that $\Phi\mathsf{HA}[\lambda]$ *is* (t', ε')*-hard and defines a* η*-regular lossy function. Then for any* $(q_h, q_g, q_{s_1}, q_{s_2})$*, RSA-sibling is* $(t, \varepsilon, q_h, q_g, q_{s_1}, q_{s_2})$*-sibling secure in the Random Oracle Model, where*

$$\begin{aligned} \varepsilon &\leqslant \left(\frac{2\eta - 1}{\eta - 1}\right) \cdot \varepsilon' + \frac{(q_g + q_f + q_{s_2})^2}{2^{\ell_{\mathtt{H}}}} \\ t' &\approx t + q_f \cdot t_{\mathsf{Enc}} + q_g \cdot t_{\mathsf{RSA}}. \end{aligned}$$

7 Conclusions

This paper presents the first security proofs for the joint use of the PSS00 and PKCS1 signatures under plausible cryptographic hardness assumptions. In particular, we prove full Sibling Unforgeability in the Random Oracle Model and give a tight security proof under the $\Phi\mathsf{HA}[\lambda]$. This matches the known security proofs of the underlying schemes, including our new proof for PSS00. Our proofs do inherit the parameter expansion from the PKCS1 proofs, but as discussed by Jager et al. [JKM18a, JKM18b], using an additional assumption, one can lift our results from our variant to the more standard two prime modulus instantiations of PKCS1 and PSS00. Thus, the current joint usage in practice is indeed secure, provided the parameters are chosen appropriately.

This paper also presents a more idealized proof, that while not in standard usage, could potentially be adopted in the future. This is given that it relies on the new SHA-3 XOFs, which one can expect to be integrated into standards in the future. If, or indeed when, this is the case, moving to the idealized case, without having the code expansion normally associated with it.

Acknowledgements

The authors would like to thank the anonymous SSR reviewers for their insightful comments. We would also like to thank Tibor Jager and Gareth T. Davies for their input and helpful corrections to the paper. Furthermore, we would like to thank Burt Kaliski for the several discussions and insights into the design choices made in the standards. This work was done while the author was employed at Paderborn University.

References

[ABBC10] Tolga Acar, Mira Belenkiy, Mihir Bellare, and David Cash. Cryptographic agility and its relation to circular encryption. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 403–422, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-13190-5_21. (Cited on page 4.)

[AMV15] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signature schemes. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 364–375, Denver, CO, USA, October 12–16, 2015. ACM Press. doi:10.1145/2810103.2813635. (Cited on page 3.)

[BDPA11] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The keccak sha-3 submission. Submission to NIST (Round 3), 2011. URL: http://keccak.noekeon.org/Keccak-submission-3.pdf. (Cited on page 23.)

[BFK+12] Romain Bardou, Riccardo Focardi, Yusuke Kawamoto, Lorenzo Simionato, Graham Steel, and Joe-Kai Tsay. Efficient padding oracle attacks on cryptographic hardware. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 608–625, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-32009-5_36. (Cited on page 2.)

[BFK+14] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Santiago Zanella Béguelin. Proving the TLS handshake secure (as it is). In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 235–255, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-44381-1_14. (Cited on page 4.)

[BJLS16] Christoph Bader, Tibor Jager, Yong Li, and Sven Schäge. On the impossibility of tight cryptographic reductions. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 273–304, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-49896-5_10. (Cited on page 3.)

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12, Santa Barbara, CA, USA, August 23–27, 1998. Springer, Heidelberg, Germany. doi:10.1007/BFb0055716. (Cited on page 2.)

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press. doi:10.1145/168588.168596. (Cited on page 3.)

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416, Saragossa, Spain, May 12–16, 1996. Springer, Heidelberg, Germany. doi:10.1007/3-540-68339-9_34. (Cited on page 2, 3, 7, 8.)

[BSY17] Hanno Böck, Juraj Somorovsky, and Craig Young. Return of bleichenbacher's oracle threat (ROBOT). Cryptology ePrint Archive, Report 2017/1189, 2017. https://eprint.iacr.org/2017/1189. (Cited on page 2.)

[BY93] Mihir Bellare and Moti Yung. Certifying cryptographic tools: The case of trapdoor permutations. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 442–460, Santa Barbara, CA, USA, August 16–20, 1993. Springer, Heidelberg, Germany. doi:10.1007/3-540-48071-4_31. (Cited on page 3.)

[BY96] Mihir Bellare and Moti Yung. Certifying permutations: Noninteractive zero-knowledge based on any trapdoor permutation. *Journal of Cryptology*, 9(3):149–166, June 1996. (Cited on page 3.)

[CDD17] Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. Practical UC-secure delegatable credentials with attributes and their application to blockchain. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 683–699, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press. doi:10.1145/3133956.3134025. (Cited on page 4, 5, 6.)

[CFPR96] Don Coppersmith, Matthew K. Franklin, Jacques Patarin, and Michael K. Reiter. Low-exponent RSA with related messages. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 1–9, Saragossa, Spain, May 12–16, 1996. Springer, Heidelberg, Germany. doi:10.1007/3-540-68339-9_1. (Cited on page 2.)

[CJNP00] Jean-Sébastien Coron, Marc Joye, David Naccache, and Pascal Paillier. New attacks on PKCS#1 v1.5 encryption. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 369–381, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany. doi:10.1007/3-540-45539-6_25. (Cited on page 2.)

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany. doi:10.1007/3-540-48910-X_28. (Cited on page 3, 7.)

[Cop97] Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, September 1997. doi:10.1007/s001459900030. (Cited on page 7.)

[Cor00] Jean-Sébastien 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, Santa Barbara, CA, USA, August 20–24, 2000. Springer, Heidelberg, Germany. doi:10.1007/3-540-44598-6_14. (Cited on page 3.)

[Cor02a] Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 272–287, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany. doi:10.1007/3-540-46035-7_18. (Cited on page 3.)

[Cor02b] Jean-Sébastien Coron. Security proof for partial-domain hash signature schemes. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 613–626, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany. doi:10.1007/3-540-45708-9_39. (Cited on page 4.)

[DLP+12] Jean Paul Degabriele, Anja Lehmann, Kenneth G. Paterson, Nigel P. Smart, and Mario Strefler. On the joint security of encryption and signature in EMV. In Orr Dunkelman, editor, *Topics in Cryptology – CT-RSA 2012*, volume 7178 of *Lecture Notes in Computer Science*, pages 116–135, San Francisco, CA, USA, February 27 – March 2, 2012. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-27954-6_8. (Cited on page 2, 4.)

[HP01] Stuart Haber and Benny Pinkas. Securely combining public-key cryptosystems. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001: 8th Conference on Computer and Communications Security*, pages 215–224, Philadelphia, PA, USA, November 5–8, 2001. ACM Press. doi:10.1145/501983.502013. (Cited on page 4.)

[JKM18a] Tibor Jager, Saqib A. Kakvi, and Alexander May. On the security of the PKCS#1 v1.5 signature scheme. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 1195–1208, Toronto, ON, Canada, October 15–19, 2018. ACM Press. doi:10.1145/3243734.3243798. (Cited on page 2, 3, 4, 16, 18, 19, 24, 29.)

[JKM18b] Tibor Jager, Saqib A. Kakvi, and Alexander May. On the security of the PKCS#1 v1.5 signature scheme. Cryptology ePrint Archive, Report 2018/855, 2018. https://eprint.iacr.org/2018/855. (Cited on page 2, 3, 4, 16, 18, 19, 24, 29.)

[Jon01] Jakob Jonsson. Security proofs for the RSA-PSS signature scheme and its variants. Cryptology ePrint Archive, Report 2001/053, 2001. http://eprint.iacr.org/2001/053. (Cited on page 2, 3, 7, 10.)

[JSS15a] Tibor Jager, Jörg Schwenk, and Juraj Somorovsky. On the security of TLS 1.3 and QUIC against weaknesses in PKCS#1 v1.5 encryption. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 1185–1196, Denver, CO, USA, October 12–16, 2015. ACM Press. doi:10.1145/2810103.2813657. (Cited on page 2.)

[JSS15b] Tibor Jager, Jörg Schwenk, and Juraj Somorovsky. Practical invalid curve attacks on TLS-ECDH. In Günther Pernul, Peter Y. A. Ryan, and Edgar R. Weippl, editors, *ESORICS 2015: 20th European Symposium on Research in Computer Security, Part I*, volume 9326 of *Lecture Notes in Computer Science*, pages 407–425, Vienna, Austria, September 21–25, 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-24174-6_21. (Cited on page 2.)

[KK12] Saqib A. Kakvi and Eike Kiltz. Optimal security proofs for full domain hash, revisited. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 537–553, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-29011-4_32. (Cited on page 3, 7, 10.)

[KK18] Saqib A. Kakvi and Eike Kiltz. Optimal security proofs for full domain hash, revisited. *Journal of Cryptology*, 31(1):276–306, January 2018. doi:10.1007/s00145-017-9257-9. (Cited on page 3, 7, 10.)

[KKM12] Saqib A. Kakvi, Eike Kiltz, and Alexander May. Certifying RSA. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 404–414, Beijing, China, December 2–6, 2012. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-34961-4_25. (Cited on page 3, 7.)

[KOS10] Eike Kiltz, Adam O'Neill, and Adam Smith. Instantiability of RSA-OAEP under chosen-plaintext attack. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 295–313, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-14623-7_16. (Cited on page 4.)

[KPR03] Vlastimil Klíma, Ondrej Pokorný, and Tomás Rosa. Attacking RSA-based sessions in SSL/TLS. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 426–440, Cologne, Germany, September 8–10, 2003. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-45238-6_33. (Cited on page 2.)

[KPS13] Eike Kiltz, Krzysztof Pietrzak, and Mario Szegedy. Digital signatures with minimal overhead from indifferentiable random invertible functions. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 571–588, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-40041-4_31. (Cited on page 4.)

[LOS13] Mark Lewko, Adam O'Neill, and Adam Smith. Regularity of lossy RSA on subdomains and its applications. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 55–75, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-38348-9_4. (Cited on page 4.)

[MSW+14] Christopher Meyer, Juraj Somorovsky, Eugen Weiss, Jörg Schwenk, Sebastian Schinzel, and Erik Tews. Revisiting SSL/TLS implementations: New bleichenbacher side channels and attacks. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 733–748. USENIX Association, 2014. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/meyer. (Cited on page 2.)

[PSST11] Kenneth G. Paterson, Jacob C. N. Schuldt, Martijn Stam, and Susan Thomson. On the joint security of encryption and signature, revisited. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 161–178, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-25385-0_9. (Cited on page 4.)

[PW08] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 187–196, Victoria, BC, Canada, May 17–20, 2008. ACM Press. doi:10.1145/1374376.1374406. (Cited on page 3.)

[Seu14] Yannick Seurin. On the lossiness of the Rabin trapdoor function. In Hugo Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 380–398, Buenos Aires, Argentina, March 26–28, 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-54631-0_22. (Cited on page 4.)

[SZ15] Adam Smith and Ye Zhang. On the regularity of lossy RSA - improved bounds and applications to padding-based encryption. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part I*, volume 9014 of *Lecture Notes in Computer Science*, pages 609–628, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-46494-6_25. (Cited on page 4.)

[ZJRR14] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-tenant side-channel attacks in PaaS clouds. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014: 21st Conference on Computer and Communications Security*, pages 990–1003, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press. doi:10.1145/2660267.2660356. (Cited on page 2.)

Standards Documents

[FIPS 202] Information Technology Laboratory and National Institute of Standards and Technology. *FIPS-202 SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, August 2015. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf. (Cited on page 24.)

[IEEE 1363] Burt Kaliski (ed.). IEEE standard specifications for public-key cryptography. *IEEE Std 1363-2000*, pages 1–228, Aug 2000. https://ieeexplore.ieee.org/servlet/opac?punumber=7168. doi:10.1109/IEEESTD.2000.92292. (Cited on page 2, 7, 8, 9, 10, 14, 19, 27.)

[ISO 9796-2] International Standards Organisation. Information technology – security techniques – digital signature schemes giving message recovery – part 2: Integer factorization based mechanisms. ISO 9796-2:2010, International Organization for Standardization, Geneva, Switzerland, 2010. https://www.iso.org/standard/54788.html. (Cited on page 4.)

[SP 800-185] John Kelsey, Shu-jen Chang, and Ray Perlner. Sha-3 derived functions: cshake, kmac, tuplehash, and parallelhash. Technical report, National Institute of Standards and Technology, 2016. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf. (Cited on page 24.)

Primary RFC References

[RFC 2246] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), January 1999. Obsoleted by [RFC 4346], updated by [RFC 3546, RFC 5746, RFC 6176, RFC 7465, RFC 7507, RFC 7919]. URL: https://www.rfc-editor.org/rfc/rfc2246.txt, doi:10.17487/RFC2246. (Cited on page 16.)

[RFC 2313] B. Kaliski. PKCS #1: RSA Encryption Version 1.5. RFC 2313 (Informational), March 1998. Obsoleted by [RFC 2437]. URL: https://www.rfc-editor.org/rfc/rfc2313.txt, doi:10.17487/RFC2313. (Cited on page 2, 16.)

[RFC 2437] B. Kaliski and J. Staddon. PKCS #1: RSA Cryptography Specifications Version 2.0. RFC 2437 (Informational), October 1998. Obsoleted by [RFC 3447]. URL: https://www.rfc-editor.org/rfc/rfc2437.txt, doi:10.17487/RFC2437. (Cited on page 2, 3, 13, 34.)

[RFC 3275] D. Eastlake 3rd, J. Reagle, and D. Solo. (Extensible Markup Language) XML-Signature Syntax and Processing. RFC 3275 (Draft Standard), March 2002. URL: https://www.rfc-editor.org/rfc/rfc3275.txt, doi:10.17487/RFC3275. (Cited on page 16.)

[RFC 3447] J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447 (Informational), February 2003. Obsoleted by [RFC 8017]. URL: https://www.rfc-editor.org/rfc/rfc3447.txt, doi:10.17487/RFC3447. (Cited on page 34.)

[RFC 3770] R. Housley and T. Moore. Certificate Extensions and Attributes Supporting Authentication in Point-to-Point Protocol (PPP) and Wireless Local Area Networks (WLAN). RFC 3770 (Proposed Standard), May 2004. Obsoleted by [RFC 4334]. URL: https://www.rfc-editor.org/rfc/rfc3770.txt, doi:10.17487/RFC3770. (Cited on page 16.)

[RFC 4055] J. Schaad, B. Kaliski, and R. Housley. Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 4055 (Proposed Standard), June 2005. Updated by [RFC 5756]. URL: https://www.rfc-editor.org/rfc/rfc4055.txt, doi:10.17487/RFC4055. (Cited on page 16.)

[RFC 4346] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard), April 2006. Obsoleted by [RFC 5246], updated by [RFC 4366, RFC 4680, RFC 4681, RFC 5746, RFC 6176, RFC 7465, RFC 7507, RFC 7919]. URL: https://www.rfc-editor.org/rfc/rfc4346.txt, doi:10.17487/RFC4346. (Cited on page 16, 34.)

[RFC 4359] B. Weis. The Use of RSA/SHA-1 Signatures within Encapsulating Security Payload (ESP) and Authentication Header (AH). RFC 4359 (Proposed Standard), January 2006. URL: https://www.rfc-editor.org/rfc/rfc4359.txt, doi:10.17487/RFC4359. (Cited on page 16.)

[RFC 4880] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. OpenPGP Message Format. RFC 4880 (Proposed Standard), November 2007. Updated by [RFC 5581]. URL: https://www.rfc-editor.org/rfc/rfc4880.txt, doi:10.17487/RFC4880. (Cited on page 16.)

[RFC 5246] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Obsoleted by [RFC 8446], updated by [RFC 5746, RFC 5878, RFC 6176, RFC 7465, RFC 7507, RFC 7568, RFC 7627, RFC 7685, RFC 7905, RFC 7919, RFC 8477]. URL: https://www.rfc-editor.org/rfc/rfc5246.txt, doi:10.17487/RFC5246. (Cited on page 16, 34, 35.)

[RFC 7515] M. Jones, J. Bradley, and N. Sakimura. JSON Web Signature (JWS). RFC 7515 (Proposed Standard), May 2015. URL: https://www.rfc-editor.org/rfc/rfc7515.txt, doi:10.17487/RFC7515. (Cited on page 16.)

[RFC 8017] K. Moriarty (Ed.), B. Kaliski, J. Jonsson, and A. Rusch. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017 (Informational), November 2016. URL: https://www.rfc-editor.org/rfc/rfc8017.txt, doi:10.17487/RFC8017. (Cited on page 2, 13, 34.)

[RFC 8446] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard), August 2018. URL: https://www.rfc-editor.org/rfc/rfc8446.txt, doi:10.17487/RFC8446. (Cited on page 2, 35, 36.)

Secondary RFC References

[RFC 3546] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. Transport Layer Security (TLS) Extensions. RFC 3546 (Proposed Standard), June 2003. Obsoleted by [RFC 4366]. URL: https://www.rfc-editor.org/rfc/rfc3546.txt, doi:10.17487/RFC3546.

[RFC 4334] R. Housley and T. Moore. Certificate Extensions and Attributes Supporting Authentication in Point-to-Point Protocol (PPP) and Wireless Local Area Networks (WLAN). RFC 4334 (Proposed Standard), February 2006. URL: https://www.rfc-editor.org/rfc/rfc4334.txt, doi:10.17487/RFC4334.

[RFC 4366] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. Transport Layer Security (TLS) Extensions. RFC 4366 (Proposed Standard), April 2006. Obsoleted by [RFC 5246, RFC 6066], updated by [RFC 5746]. URL: https://www.rfc-editor.org/rfc/rfc4366.txt, doi:10.17487/RFC4366.

[RFC 4680] S. Santesson. TLS Handshake Message for Supplemental Data. RFC 4680 (Proposed Standard), October 2006. Updated by [RFC 8477]. URL: https://www.rfc-editor.org/rfc/rfc4680.txt, doi:10.17487/RFC4680.

[RFC 4681] S. Santesson, A. Medvinsky, and J. Ball. TLS User Mapping Extension. RFC 4681 (Proposed Standard), October 2006. URL: https://www.rfc-editor.org/rfc/rfc4681.txt, doi:10.17487/RFC4681.

[RFC 5746] E. Rescorla, M. Ray, S. Dispensa, and N. Oskov. Transport Layer Security (TLS) Renegotiation Indication Extension. RFC 5746 (Proposed Standard), February 2010. URL: https://www.rfc-editor.org/rfc/rfc5746.txt, doi:10.17487/RFC5746.

[RFC 5756] S. Turner, D. Brown, K. Yiu, R. Housley, and T. Polk. Updates for RSAES-OAEP and RSASSA-PSS Algorithm Parameters. RFC 5756 (Proposed Standard), January 2010. URL: https://www.rfc-editor.org/rfc/rfc5756.txt, doi:10.17487/RFC5756.

[RFC 5581] D. Shaw. The Camellia Cipher in OpenPGP. RFC 5581 (Informational), June 2009. URL: https://www.rfc-editor.org/rfc/rfc5581.txt, doi:10.17487/RFC5581.

[RFC 5878] M. Brown and R. Housley. Transport Layer Security (TLS) Authorization Extensions. RFC 5878 (Experimental), May 2010. Updated by [RFC 8477]. URL: https://www.rfc-editor.org/rfc/rfc5878.txt, doi:10.17487/RFC5878.

[RFC 6066] D. Eastlake 3rd. Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066 (Proposed Standard), January 2011. Updated by [RFC 8446, RFC 8449]. URL: https://www.rfc-editor.org/rfc/rfc6066.txt, doi:10.17487/RFC6066.

[RFC 6176] S. Turner and T. Polk. Prohibiting Secure Sockets Layer (SSL) Version 2.0. RFC 6176 (Proposed Standard), March 2011. URL: https://www.rfc-editor.org/rfc/rfc6176.txt, doi:10.17487/RFC6176.

[RFC 7465] A. Popov. Prohibiting RC4 Cipher Suites. RFC 7465 (Proposed Standard), February 2015. URL: https://www.rfc-editor.org/rfc/rfc7465.txt, doi:10.17487/RFC7465.

[RFC 7507] B. Moeller and A. Langley. TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks. RFC 7507 (Proposed Standard), April 2015. URL: https://www.rfc-editor.org/rfc/rfc7507.txt, doi:10.17487/RFC7507.

[RFC 7568] R. Barnes, M. Thomson, A. Pironti, and A. Langley. Deprecating Secure Sockets Layer Version 3.0. RFC 7568 (Proposed Standard), June 2015. URL: https://www.rfc-editor.org/rfc/rfc7568.txt, doi:10.17487/RFC7568.

[RFC 7627] K. Bhargavan (Ed.), A. Delignat-Lavaud, A. Pironti, A. Langley, and M. Ray. Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension. RFC 7627 (Proposed Standard), September 2015. URL: https://www.rfc-editor.org/rfc/rfc7627.txt, doi:10.17487/RFC7627.

[RFC 7685] A. Langley. A Transport Layer Security (TLS) ClientHello Padding Extension. RFC 7685 (Proposed Standard), October 2015. URL: https://www.rfc-editor.org/rfc/rfc7685.txt, doi:10.17487/RFC7685.

[RFC 7905] A. Langley, W. Chang, N. Mavrogiannopoulos, J. Strombergson, and S. Josefsson. ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS). RFC 7905 (Proposed Standard), June 2016. URL: https://www.rfc-editor.org/rfc/rfc7905.txt, doi:10.17487/RFC7905.

[RFC 7919] D. Gillmor. Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS). RFC 7919 (Proposed Standard), August 2016. URL: https://www.rfc-editor.org/rfc/rfc7919.txt, doi:10.17487/RFC7919.

[RFC 8477] J. Salowey and S. Turner. IANA Registry Updates for TLS and DTLS. RFC 8447 (Proposed Standard), August 2018. URL: https://www.rfc-editor.org/rfc/rfc8447.txt, doi:10.17487/RFC8447.

[RFC 8449] M. Thomson. Record Size Limit Extension for TLS. RFC 8449 (Proposed Standard), August 2018. URL: https://www.rfc-editor.org/rfc/rfc8449.txt, doi:10.17487/RFC8449.