Security Analysis of RSA-BSSA*

Anna Lysyanskaya

Brown University
Providence, RI 02912, USA
anna_lysyanskaya@brown.edu

Abstract. In a blind signature scheme, a user can obtain a digital signature on a message of her choice without revealing anything about the message or the resulting signature to the signer. Blind signature schemes have recently found applications for privacy-preserving web browsing and ad ecosystems, and as such, are ripe for standardization. In this paper, we show that the recent proposed standard of Denis, Jacobs and Wood [18, 17] constitutes a strongly one-more-unforgeable blind signature scheme in the random-oracle model under the one-more-RSA assumption. Further, we show that the blind version of RSA-FDH proposed and analyzed by Bellare, Namprempre, Pointcheval and Semanko [6] does not satisfy blindness when the public key is chosen maliciously, but satisfies a weaker notion of a blind token.

1 Introduction

A blind signature scheme is a digital signature scheme that allows the signature recipient to obtain a digital signature on a message of the recipient's choice without revealing this message to the signer. The key feature of a blind signature protocol is that the resulting signature cannot be linked to a particular protocol run. If the recipient ran the protocol n times and, as a result, produced n signatures and provided them to the signer in a randomly permuted order, the signer would not be able to identify which signature corresponded to which protocol run any better than by guessing at random. Just as in a regular digital signature scheme, in order to verify a signature, a verifier (a third party, distinct from the signer or the signature recipient) runs a non-interactive verification algorithm.

Applications. Blind signatures were first introduced by David Chaum [13, 14]. The motivating application was untraceable electronic cash (ecash) [13, 15]: a bank can issue electronic coins by issuing blind signatures. A message represents a coin's serial number, while the bank's signature on it attests that it is indeed a valid coin. The fact that it was issued via a blind signing protocol means that one cannot trace which coin was issued to which user, and therefore cannot surmise how a particular user Alice spent her money.

Blind signatures protect a user's privacy even while ensuring they are qualified for a particular transaction. For example, suppose that a user has convinced

* A shorter version of this paper will appear in PKC 2023.

a server that he is a human (rather than a bot) by solving a CAPTCHA. Then the server may issue such a user a blind signature (or several blind signatures) that allow this user to convince other servers that he is a human and not a bot without needing to perform additional CAPTCHAs; however, even if all these servers compare transaction logs, they cannot tell which user it was. This simple scenario is of a growing importance in practice; for example, in is used in VPN by Google One[1], Apple's iCloud Private Relay[2] and Apple's Safari browser proposal for privacy-preserving click measurements[3].

Definitions. A blind signature scheme must satisfy correctness, blindness, and strong one-more unforgeability [30, 26, 1, 35]. Correctness means that an honest verifier will always accept a signature issued by an honest signer to an honest recipient; here, by "honest" we mean one that follows the prescribed algorithms. Blindness, as we explained above, means that the malicious signer learns nothing about a message during the signing protocol, and a signature cannot be linked to the specific protocol execution in which it was computed. This must hold even if the signer's public key is chosen maliciously. Finally, strong one-more unforgeability means that, if an adversary acts as the recipient n times, then it cannot produce $n+1$ distinct message-signature pairs better than with negligible probability. It is important that unforgeability hold even when the adversary engages in several sessions with the signer at the same time; i.e. it is important that unforgeability should hold in the *concurrent* setting.

Standardization. Blind signatures have been studied for almost forty years. They have well-understood definitions of security [30, 26, 1, 35]. Numerous constructions have also been proposed [13, 29, 30, 5, 6, 3, 2, 11, 20, 24, 25]. Finally, as we argued above, they are highly desirable in practice. Of course, even a well-understood cryptographic primitive should not get adopted for widespread use without undergoing a thorough standardization process through software standardization bodies such as the IETF.

The first proposed IETF standard for a blind signature scheme is the RSA-BSSA proposal by Denis, Jacobs and Wood [18, 17]. The scheme they proposed for standardization is, in a nutshell, the blind version of RSA-PSS [8, 9, 31, 32] along the lines proposed by Chaum [13, 14]. However, as the analysis in this paper makes clear, care must be taken to ensure that the message being signed comes from a high-entropy distribution; in the event that it doesn't, a random salt value must be appended to it.

The key generation and verification algorithms are (essentially) the same as in RSA-PSS, except that, in case the message `msg` does not come from a high-entropy distribution, a salt value `rand` must be concatenated to the message `msg`. More precisely, if `msg` does not come from a high-entropy distribution, this paper's analysis recommends that the blind signing algorithm consist of three steps: first, on input a message `msg` and the RSA public key (N, e), the user chooses a random salt value `rand` and computes an RSA-PSS encoding m of

[1] https://one.google.com/about/vpn/howitworks

[2] https://www.apple.com/privacy/docs/iCloud_Private_Relay_Overview_Dec2021.PDF

[3] https://webkit.org/blog/11940/pcm-click-fraud-prevention-and-attribution-sent-to-advertiser/

`msg` $\circ$ `rand` (where '$\circ$' denotes concatenation), picks a blinding value r and sends the value $z = mr^e \bmod N$ to the signer. Using his secret key d, the signer computes $s = z^d \bmod N$ and sends it to the user, who derives the signature $\sigma = s/r \bmod N$; it is easy to see that $\sigma^e = s^e/r^e = z/r^e = m \bmod N$, and thus, constitutes a valid RSA-PSS signature on the user's message `msg` $\circ$ `rand`. In case `msg` comes from a high-entropy distribution, `rand` is not needed, and m is computed as a PSS encoding of `msg`; the rest of the signing algorithm is the same. As we will see in Section 4, either the high entropy of `msg`, or the additional salt value `rand` are necessary to ensure that the scheme is provably blind in the event that the signer's key was chosen maliciously. This has resulted in IETF discussions on amending the draft[4].

As pointed out by Denis, Jacobs and Wood [18, 17], the message-response (i.e., two-move) structure of this protocol makes it desirable. The security game for strong one-more unforgeability for a two-move protocol is the same whether in the sequential or the concurrent setting. In contrast, a recent result [10] gave an attack on popular three-move blind signature protocols (such as the blind version of the Schnorr signature [33, 30, 34] or anonymous credentials light [4]) in the concurrent setting, making them poor candidates for standardization. Moreover, a three-message (or more) protocol would require the signer to keep state, which is a significant complication when it comes to system design, making the concurrently secure blind signature schemes of Abe [2] and Tessaro and Zhu [37] less suitable in practice.

The choice of blind RSA-PSS over blind RSA-FDH [6] is motivated by the popularity of (non-blind) RSA-PSS, ensuring that, at least as far as verifying the signatures is concerned, no new software need be developed. That way, even unsophisticated participants have easy access to the digital tools they need to take advantage of the privacy-preserving features offered by blind signatures.

Why standardize and adopt an RSA-based scheme now, instead of a post-quantum one? Indeed it is possible that, with the advent of quantum computing, decades from now another scheme will have to replace this RSA-based one. Yet, this will have no consequences on today's clients and servers if the users' privacy is protected even from quantum computers (for example, if it holds unconditionally). The consequences to the servers are minimized because a blind signature ceases to be relevant after a relatively brief amount of time, so the lifetime of a signing key would be measured in weeks rather than years.

This paper's contributions and organization. We show that the proposed RSA-BSSA standard [17] constitutes a one-more unforgeable blind signature scheme. One-more unforgeability holds in the random-oracle model under the one-more-RSA assumption introduced by Bellare, Namprempre, Pointcheval and Semanko (BNPS) [6]. Blindness of the RSA-BSSA holds in the random-oracle model.

We also show that Chaum-BNPS's blind RSA-FDH [14, 6] is not blind in the malicious-signer model, i.e., it can only be shown to be blind if the signer's key pair is generated honestly (see Section 4.5). However, we show in Section 4.5

[4] https://github.com/cfrg/draft-irtf-cfrg-blind-signatures/pull/105

that even in the case of a malicious signer, it satisfies the weaker notion of a blind token which we introduce in Section 2.3.

The rest of this paper is organized as follows: In Section 2 we recall the definition of security for blind signature schemes. Our definitions are tailor-made for two-move blind signature schemes, because in the case of two-move signatures the issues of composition with other protocols go away (as discussed above). Other than that, our definitions are standard [30, 35, 26, 1]. We include bibliographic notes explaining that at the end of Sections 2.1 and 2.2 that provide definitions of one-more unforgeability and blindness, respectively.

In Section 3 we give an overview of RSA-BSSA. We begin by giving a basic version of the scheme, in which the blind signature that a user obtains is a standard RSA-PSS signature on the user's message `msg` (i.e. there is no `rand`). We also give two modifications of the basic scheme: a variant in which the signer's RSA public key (N, e) is enhanced in a way that ensures that the exponent e is relatively prime to $\varphi(N)$ using a technique of Goldberg, Reyzin, Sagga and Baldimtsi [22]. Finally, in Section 3.3 we give the variant that corresponds to the RSA-BSSA proposal from February 2022 [17]; in this variant, the public key is a standard RSA public key (N, e) and the signature on a message `msg` consists of a salt `rand` and the PSS signature on $(\texttt{msg} \circ \texttt{rand})$.

In Section 4 we justify the salt `rand`: we show why it is difficult to prove that the basic scheme is blind without introducing additional assumptions, and show that, in the random-oracle model, both modifications give rise to blind signature schemes. We also show that the basic scheme is a blind token. Finally, in Section 5 we show that the basic scheme and both variants are one-more-unforgeable under the one-more-RSA assumption in the random-oracle model.

2 Definition of a two-move blind signature scheme

The definition of a blind signature scheme we provide here applies only to two-move blind signatures; see prior work for more general definitions [8, 9, 31, 32]. First, in Definition 1 let us give the input-output specification for the five algorithms that constitute a two-move blind signature scheme. The key generation algorithm KeyGen and the signature verification algorithm Verify have the same input-output behavior as in a regular digital signature scheme.

The signing algorithm is broken down into three steps: (1) The signature recipient runs the Blind algorithm to transforms a message `msg` into its blinded form `blinded_msg`; `blinded_msg` is sent to the signer. (2) The signer runs the algorithm $\mathsf{BSig}(SK, \texttt{blinded\_msg})$ to compute its response `blinded_sig`, and then sends it to the signature recipient. (3) The signature recipient uses the algorithm Finalize to transform `blinded_sig` into a valid signature σ on its message `msg`. More precisely:

Definition 1 (Input-output specification for a two-move blind signature scheme). *Let* $\mathcal{S} = (\mathsf{KeyGen}, \mathsf{Blind}, \mathsf{BSig}, \mathsf{Finalize}, \mathsf{Verify})$ *be a set of polynomial-time algorithms with the following input-output specifications:*

$\mathsf{KeyGen}(1^k) \rightarrow (PK, SK)$ *is a probabilistic algorithm that takes as input* 1^k *(the security parameter represented in unary) and outputs the public signature verification key* PK *and a secret signing key* SK*.*

$\mathsf{Blind}(PK, \mathtt{msg}) \rightarrow (\mathtt{blinded\_msg}, \mathtt{inv})$ *is a probabilistic algorithm that takes as input the public key* PK *and a string* `msg` *and outputs a blinded message* `blinded_msg` *(which will be sent to the signer) and an auxiliary string* `inv` *(which will be used by* Finalize *to derive the final signature* σ*).*

$\mathsf{BSig}(SK, \mathtt{blinded\_msg}) \rightarrow \mathtt{blinded\_sig}$ *is an algorithm (possibly a probabilistic one) that takes as input the secret signing key* SK *and a string* `blinded_msg` *and outputs a blinded signature* `blinded_sig`.

$\mathsf{Finalize}(PK, \mathtt{inv}, \mathtt{blinded\_sig}) \rightarrow \sigma$ *is an algorithm that takes as input the public signature verification key* PK*, an auxiliary string* `inv` *and a blinded signature and outputs a signature* σ*.*

$\mathsf{Verify}(PK, \mathtt{msg}, \sigma)$ *is an algorithm that either accepts or rejects.*

Next, let us define what it means for $\mathcal{S}$ to constitute a *correct* blind signature scheme. On a high level, correctness means that if a signature σ was produced after both the signature recipient and the signer followed their corresponding algorithms, then this signature will be accepted by Verify. More formally:

Definition 2 (Correct two-move blind signature). *Let* $\mathcal{S} = (\mathsf{KeyGen}, \mathsf{Blind}, \mathsf{BSig}, \mathsf{Finalize}, \mathsf{Verify})$ *be a set of polynomial-time algorithms that satisfy the input-output specification for a two-move blind signature scheme (Definition 1).* $\mathcal{S}$ *constitutes a correct two-move blind signature scheme if for all* k*,* (PK, SK) *output by* $\mathsf{KeyGen}(1^k)$*, strings* `msg`*,* $(\mathtt{blinded\_msg}, \mathtt{inv})$ *output by* $\mathsf{Blind}(PK, \mathtt{msg})$*,* `blinded_sig` *output by* $\mathsf{BSig}(SK, \mathtt{blinded\_msg})$*, and* σ *output by* $\mathsf{Finalize}(PK, \mathtt{inv}, \mathtt{blinded\_sig})$*,* $\mathsf{Verify}(PK, \mathtt{msg}, \sigma)$ *accepts.*

2.1 Strong one-more unforgeability

As discussed above, a blind signature scheme must satisfy one-more unforgeability: an adversarial user who obtained ℓ signatures from the signer cannot produce $\ell + 1$ distinct message-signature pairs. Since we are limiting our attention to two-move blind signatures, the security experiment that captures it can allow the adversary oracle access to the algorithm $\mathsf{BSig}(SK, \cdot)$. More formally:

Definition 3 (One-more-unforgeability). *Let* $\mathcal{S} = \mathsf{KeyGen}, \mathsf{Blind}, \mathsf{BSig}, \mathsf{Finalize}, \mathsf{Verify}$ *be a set of polynomial-time algorithms that satisfy the input-output specification for a two-move blind signature scheme (Definition 1).*

For an algorithm $\mathcal{A}$ *(modeled as an oracle Turing machine), the success probability* $p^{\mathcal{S}}_{\mathcal{A}}(k)$ *of* $\mathcal{A}$ *in breaking the one-more-unforgeability of* $\mathcal{S}$ *is the probability that* $\mathcal{A}$ *is successful in the following experiment parameterized by* k*:*

Experiment set-up *The key pair is generated:* $(PK, SK) \leftarrow \mathsf{KeyGen}(1^k)$*.*

Adversary's execution *The adversary $\mathcal{A}$ is given oracle access to* $\mathsf{BSig}(SK,\cdot)$ *and is run on input PK;* $\mathcal{A}^{\mathsf{BSig}(SK,\cdot)}(PK)$ *terminates with a set of message-signature pairs on its output tape:* $((\mathtt{msg}_1,\sigma_1),\ldots,(\mathtt{msg}_n,\sigma_n))$*, and a set of query-response pairs on its query tape:*

$$((\mathtt{blinded\_msg}_1,\mathtt{blinded\_sig}_1),\ldots,(\mathtt{blinded\_msg}_\ell,\mathtt{blinded\_sig}_\ell))$$

The success criterion *The number of distinct messages* $\mathtt{msg}_i$ *such that* $\mathsf{Verify}(PK,\mathtt{msg}_i,\sigma_i)=1$ *is at least* $\ell+1$*, i.e.* $\mathcal{A}$ *output more signatures on distinct messages than the number of queries it made to* BSig*.*

The success criterion for strong one-more-unforgeability *The number of distinct message-signature pairs* $(\mathtt{msg}_i,\sigma_i)$ *such that* $\mathsf{Verify}(PK,\mathtt{msg}_i,\sigma_i)=1$ *is at least* $\ell+1$*, i.e.* $\mathcal{A}$ *output more distinct signatures than the number of queries it made to* BSig*.*

$\mathcal{S}$ satisfies the one-more-unforgeability property if for any polynomial-time adversary $\mathcal{A}$, the value $p^{\mathcal{S}}_{\mathcal{A}}(k)$ is negligible.

Definition 4 (Strong one-more-unforgeability). *Let* $\mathcal{S}=(\mathsf{KeyGen},\mathsf{Blind},\mathsf{BSig},\mathsf{Finalize},\mathsf{Verify})$ *be a set of polynomial-time algorithms that satisfy the input-output specification for a two-move blind signature scheme (Definition 1). For an oracle Turing machine $\mathcal{A}$, the success probability $p^{\mathcal{S}}_{\mathcal{A}}(k)$ of $\mathcal{A}$ in breaking the strong one-more unforgeability of $\mathcal{S}$ is the probability that $\mathcal{A}$ is successful in the following experiment parameterized by k:*

Experiment set-up *The key pair is generated:* $(PK,SK)\leftarrow\mathsf{KeyGen}(1^k)$*.*

Adversary's execution *The adversary $\mathcal{A}$ is given oracle access to* $\mathsf{BSig}(SK,\cdot)$ *and is run on input PK;* $\mathcal{A}^{\mathsf{BSig}(SK,\cdot)}(PK)$ *terminates with a set of message-signature pairs on its output tape:* $((\mathtt{msg}_1,\sigma_1),\ldots,(\mathtt{msg}_n,\sigma_n))$*, and a set of query-response pairs on its query tape:*

$$((\mathtt{blinded\_msg}_1,\mathtt{blinded\_sig}_1),\ldots,(\mathtt{blinded\_msg}_\ell,\mathtt{blinded\_sig}_\ell)).$$

The success criterion *The number of distinct message-signature pairs* $(\mathtt{msg}_i,\sigma_i)$ *such that* $\mathsf{Verify}(PK,\mathtt{msg}_i,\sigma_i)=1$ *is at least* $\ell+1$*, i.e.* $\mathcal{A}$ *outputs more distinct signatures than the number of queries it made to* BSig*.*

$\mathcal{S}$ satisfies the strong one-more-unforgeability property if for any polynomial-time adversary $\mathcal{A}$, the value $p^{\mathcal{S}}_{\mathcal{A}}(k)$ is negligible.

The history of this definition. Chaum's original blind signatures papers [13, 14] did not contain a formal definition; in fact, they preceded the formal definition of security for a digital signature scheme.

The regular definition of unforgeability for digital signature schemes [23] does not apply to blind signatures. In the regular definition, the adversary wins the unforgeability game if it produces a signature on a message that the challenger never signed. However, the challenger in the blind signature game has no way of knowing which messages it has signed — that's the whole point of blindness, and ideally, we want it to hold unconditionally.

Thus, Pointcheval and Stern [29, 30] came up with the notion of *one-more unforgeability* in which the adversary is considered successful if it outputs more distinct signed messages than the number of blind signing sessions it participated in. Pointcheval and Stern considered a more general structure of a blind signing protocol, not just the message-response exchange a-la our Blind, BSig, Finalize structure, and thus the issue of self-composition (i.e. what happened if the messages from the signer were adversarially interleaved with those of the adversarial users) needed to be carefully defined in their work. But, as Bellare et al. observed [6], for a protocol that has this simple two-move (i.e. message-response) structure, self-composition is for free, and so the one-more-unforgeability game can be formalized in relatively simple terms.

A stronger definition of unforgeability for blind signatures was given by Schröder and Unruh [35]. They consider the case when the adversary observes the inputs and outputs of *honest* users who engage in ℓ blind signing protocols to obtain signatures on fewer than ℓ distinct messages (i.e. some message is getting signed more than once). The adversary should not be able to get a signature on an additional message by directing honest users to get more than one signature on the same message. Schröder and Unruh showed that Pointcheval and Stern's one-more-unforgeability definition (our Definition 3) is not sufficient to prevent the adversary from taking advantage of honest users this way; but strong one-more unforgeability is. Following their work, strong one-more unforgeability is the standard notion of unforgeability for blind signature schemes.

Our formulation of strong one-more unforgeability in Definition 4 uses Definition 6.1 of Bellare et al. [6], which is their definition of one-more unforgeability, as a starting point. Their formulation is tailored specifically to one-more unforgeability of the blind RSA-FDH, while ours generally applies to any two-move protocol consisting of Blind, BSig, and Finalize. We also modified the success criterion to correspond to strong one-more unforgeability.

One might wonder why the security game is for only one signer. Indeed, we could extend the game to require that the adversary specify a number of signers and interact with each signer before outputting a set of message-signature pairs. The adversary would be deemed successful if, for one of the signers, the number of valid message-signature pairs from this signer produced by the adversary was greater than the number of the adversary's queries to this signer. It is easy to see that extending the security game to such a multi-signer scenario would not make the definition stronger: a scheme that satisfies one-more unforgeability with one signer will also satisfy it with multiple (say, n) signers. The reduction would randomly pick one of the signers and would set up the game so that it knows the secret key of all but the selected signer; the selected signer is the one from the one-more-unforgeability challenger with one signer. If the adversary succeeds and the reduction guessed the signer correctly, then the reduction will succeed as well; since the guess is correct with probability $1/n$, this shows that the two definitions are equivalent up to a security loss of $1/n$. Although not addressed explicitly in the literature cited above, this is well-understood in the context of regular digital signatures [21] and thus it is the single-signer definitions that are standard in the blind signatures literature.

2.2 Blindness

Finally, a blind signature scheme must satisfy *blindness*, that is, it should be impossible to determine which query to the (adversarial) signer resulted in the (honest) signature recipient deriving a particular message-signature pair. For this security game, the adversary picks the public key adversarially; it also picks two messages whose signatures the challenger will try to obtain. The challenger will try to obtain signatures on these messages in random order selected by picking a random bit b; the adversary's goal is to tell in what order. The adversary gets to see the resulting signatures before producing an output.

A trivial strategy for the adversary would be to issue a valid signature in response to one of the queries but not the other. In order to rule out this strategy, the challenger allows the adversary to see the resulting signatures only if both of them verify. If one (or both) of the signatures does not verify, the adversary will have to guess the bit b based on its view of the interaction with the user in the blind signing protocol.

The formal definition below applies only to two-move blind signature schemes, but it can be generalized to any protocol structure.

Definition 5 (Blindness). *Let* $\mathcal{S} = (\mathsf{KeyGen}, \mathsf{Blind}, \mathsf{BSig}, \mathsf{Finalize}, \mathsf{Verify})$ *be a set of polynomial-time algorithms that satisfy the input-output specification for a two-move blind signature scheme (Definition 1). For an interactive algorithm* $\mathcal{A}$*, let* $q^{\mathcal{S}}_{\mathcal{A}}(k, b)$ *be the probability that* $\mathcal{A}$ *outputs 0 in the following experiment parameterized by* k *and the bit* b*:*

$\mathcal{A}$ is invoked $\mathcal{A}(1^k)$ *selects a public key PK (whose length is appropriate for the security parameter* k*) and two messages* $\mathtt{msg}_0$ *and* $\mathtt{msg}_1$*.*

$\mathcal{A}$ acts as the blind signer *For* $i \in \{0, 1\}$*, the challenger computes the values* $(\mathtt{blinded\_msg}_i, \mathtt{inv}_i) \leftarrow \mathsf{Blind}(PK, \mathtt{msg}_i)$ *and sends* $(\mathtt{blinded\_msg}_b, \mathtt{blinded\_msg}_{1-b})$ *to* $\mathcal{A}$*, receiving* $(\mathtt{blinded\_sig}_b, \mathtt{blinded\_sig}_{1-b})$ *in response.*

$\mathcal{A}$ receives the signatures *For* $i \in \{0, 1\}$*, the challenger computes*

$$\sigma_i = \mathsf{Finalize}(PK, \mathtt{inv}_i, \mathtt{blinded\_sig}_i)$$

If $\mathsf{Verify}(PK, \mathtt{msg}_0, \sigma_0) = \mathsf{Verify}(PK, \mathtt{msg}_1, \sigma_1) = 1$*, it sends* (σ_0, σ_1) *to* $\mathcal{A}$*; else it sends* $\perp$ *to* $\mathcal{A}$*.*

$\mathcal{A}$'s output $\mathcal{A}$ *outputs some value* $\mathtt{output}$*.*

$\mathcal{A}$*'s advantage* $\mathbf{Adv}^{\mathcal{S}}_{\mathcal{A}}(k)$ *in breaking the blindness of* $\mathcal{S}$ *is defined as* $\mathbf{Adv}^{\mathcal{S}}_{\mathcal{A}}(k) := |q^{\mathcal{S}}_{\mathcal{A}}(k, 0) - q^{\mathcal{S}}_{\mathcal{A}}(k, 1)|$*.* $\mathcal{S}$ *satisfies blindness if for any probabilistic polynomial-time* $\mathcal{A}$*,* $\mathbf{Adv}^{\mathcal{S}}_{\mathcal{A}}(k)$ *is negligible.*

The history of this definition. The first formalization of the blindness property of a digital signature scheme was given by Juels, Luby and Ostrovsky [26]; in this initial formulation, the public key for the scheme was generated honestly. Abdalla, Namprepre and Neven [1] improved the definition by considering a signer who is already adversarial at key generation time; they also gave a more careful treatment of the compositional issues. The definition given above corresponds to the Abdalla et al. version of the blindness definition as it applies to the case of a two-move signing protocol. It is considered standard in the literature.

Again, one might wonder why the number of messages in the security game is limited to just two, $\mathtt{msg}_0$ and $\mathtt{msg}_1$; and why the user just interacts with the signer $\mathcal{A}$ once. It is relatively straightforward to show that extending the definition to allow more than two messages or to give the signer more chances to interact with the challenger will not strengthen the definition: a reduction playing middleman between the multi-message or multi-interaction adversary and the two-message single interaction challenger will inherit a non-negligible fraction of the adversary's advantage.

2.3 A new definition: Blind tokens

In certain applications, the messages being signed are chosen at random from some message space $\mathcal{M}$. If all goes well during the signing protocol, the user gets a unique authenticated token, i.e. a signature on this random message. This token should be *blind*, i.e. unlinkable to the specific interaction with the signer in which it was obtained. If for some reason the signing protocol fails to return a valid signature on this message, the message may be discarded.

Let us formalize the blindness requirement of such applications by introducing a new cryptographic primitive: a *blind token* scheme. A blind token scheme will have the same input-output specification as a blind signature scheme, and must also be strongly one-more unforgeable; however, the notion of blindness it needs to satisfy is somewhat weaker. Unlike the blind signature blindness experiment, here the two messages $\mathtt{msg}_0$ and $\mathtt{msg}_1$ are picked from the same distribution $\mathcal{M}$. The adversary has some influence on how they are picked: $\mathcal{M}$ takes as input the adversary's public key PK as well as some auxiliary input aux.

Definition 6 ((Strongly) Unforgeable blind token scheme). *Let* $\mathcal{S} =$ $\mathsf{KeyGen}, \mathsf{Blind}, \mathsf{BSig}, \mathsf{Finalize}, \mathsf{Verify}$ *be a set of polynomial-time algorithms that satisfy the input-output specification for a two-move blind signature scheme (Definition 1) and the (strong) one-more unforgeability property (Definition 3 or Definition 4). Let* $\mathcal{M}$ *be a message sampling algorithm that, on input the security parameter* 1^k*, a public key* PK*, and auxiliary input* aux*, outputs a string* $\mathtt{msg}$.

For an interactive algorithm $\mathcal{A}$ *and an efficient message sampling algorithm* $\mathcal{M}$*, let* $q_{\mathcal{A}}^{\mathcal{S},\mathcal{M}}(k,b)$ *be the probability that* $\mathcal{A}$ *outputs 0 in the following experiment parameterized by the security parameter* k *and the bit* b*:*

$\mathcal{A}$ is invoked $\mathcal{A}(1^k)$ *selects a public key* PK *(whose length is appropriate for the security parameter* k*), and auxiliary input* aux *for the message sampling algorithm.*

$\mathcal{A}$ acts as the blind signer *For $i \in \{0,1\}$, let $\texttt{msg}_i \leftarrow \mathcal{M}(1^k, PK, aux)$ be messages randomly selected by the challenger, who then proceeds to compute the values* $(\texttt{blinded\_msg}_i, \texttt{inv}_i) \leftarrow \mathsf{Blind}(PK, \texttt{msg}_i)$ *and send* $(\texttt{blinded\_msg}_b, \texttt{blinded\_msg}_{1-b})$ *to $\mathcal{A}$, receiving* $(\texttt{blinded\_sig}_b, \texttt{blinded\_sig}_{1-b})$ *in response.*

$\mathcal{A}$ receives the signatures *For $i \in \{0,1\}$, the challenger computes*

$$\sigma_i = \mathsf{Finalize}(PK, \texttt{inv}_i, \texttt{blinded\_sig}_i)$$

If $\mathsf{Verify}(PK, \texttt{msg}_0, \sigma_0) = \mathsf{Verify}(PK, \texttt{msg}_1, \sigma_1) = 1$*, it sends* $(\texttt{msg}_0, \sigma_0, \texttt{msg}_1, \sigma_1)$ *to $\mathcal{A}$; else it sends $\perp$ to $\mathcal{A}$.*

$\mathcal{A}$'s output *$\mathcal{A}$ outputs some value* $\textit{\texttt{output}}$.

$\mathcal{A}$'s advantage $\mathbf{Adv}^{\mathcal{S},\mathcal{M}}_{\mathcal{A}}(k)$ *is defined as* $\mathbf{Adv}^{\mathcal{S},\mathcal{M}}_{\mathcal{A}}(k) := |q^{\mathcal{S},\mathcal{M}}_{\mathcal{A}}(k,0) - q^{\mathcal{S},\mathcal{M}}_{\mathcal{A}}(k,1)|$. *$\mathcal{S}$ is a strongly unforgeable blind token scheme for message space $\mathcal{M}$ if for any probabilistic polynomial-time $\mathcal{A}$,* $\mathbf{Adv}^{\mathcal{S},\mathcal{M}}_{\mathcal{A}}(k)$ *is negligible.*

The motivation for this definition. This definition is new; generally, when analyzing proposed standards, introducing new notions of security is a bad idea. An algorithm adapted for practical use should satisfy a notion of security that is well-understood and established. Unfortunately, as we will see in Section 4.5, at least one scheme that is already used in practice does not satisfy the established definition of a blind signature scheme; however, we show that it satisfies Definition 6, and therefore can still be used securely in some limited applications.

In the ecash application as originally envisioned by Chaum, the message $\texttt{msg}$ is simply a string that is sampled uniformly at random; it should be long enough that it is unlikely that the same string can be sampled twice. Once the user obtains the signature σ on $\texttt{msg}$, the pair $(\texttt{msg}, \sigma)$ can be viewed as an e-coin. $\texttt{msg}$ is the coin's serial number, while σ can be thought of as its proof of validity. However, if the user fails to obtain σ for this $\texttt{msg}$, then $\texttt{msg}$ has no value and can be discarded. The reason that blind tokens give users of this system the desired privacy is that each user draws the serial numbers for her coins from exactly the same distribution as all the other users.

3 The RSA-BSSA scheme

Let us review the blind signature scheme from the RSA blind signature scheme with appendix (RSA-BSSA) proposal by Denis, Jacobs and Wood [18, 17].

High-level description of the basic scheme. In the RSA-PSS signature scheme [8, 9, 31, 32], the signature on a message M is the RSA inverse of a special encoding (called the *PSS encoding*) m of M. At a high level, the basic version of RSA-BSSA is reminiscent of Chaum's original blind signature scheme: it is the blind version of the RSA-PSS signature scheme. Following RSA-PSS, the key generation algorithm generates an RSA key pair $PK = (N, e)$, $SK = d$, where $ed \equiv 1 \bmod \varphi(N)$. Following Chaum, in order to obtain a blind signature on a message M, the user first generates a PSS encoding m of M, then blinds it using a random $r \leftarrow \mathbb{Z}_N^*$ obtaining $z = mr^e \bmod N$, which is (hopefully) an element of $\mathbb{Z}_N^*$ that is distributed independently of M. Then he gets from the signer the blinded signature $y = z^d \bmod N$, and unblinds it to obtain and output $s = yr^{-1} \bmod N$. To verify a signature s on a message M, follow the same algorithm as RSA-PSS verification: check that the PSS decoding of $m = s^e \bmod N$ is the message M. Let us fill in the missing details.

Hash functions. For the PSS encoding, the scheme will use two cryptographic hash functions Hash and MGF the same way that PSS does. Both Hash and MGF take as input a string of bytes S and an integer ℓ, and output a string of ℓ bytes. In the security analysis, both will be treated as random oracles. Even though their input-output specifications and security requirements match, it may be helpful to have functions with different implementations because, as their names suggest, the function Hash will potentially take a long string S and output a shorter string; while MGF (which stands for "mask generation function") will take as input a short "seed" string and output a longer one.

Other subroutines. Since we are analyzing not just an algorithm but a proposed standard, it is important to note that any software program implementing this standard will have to recognize two distinct types: integers (on which integer operations are performed) and strings of bytes (that lend themselves to string operations, such as concatenation and exclusive-or). $\mathsf{I2OSP}$ is a procedure that converts an integer into an octet string (an octet is just the IETF terminology for the eight-bit byte). On input an integer and the desired length ℓ, it outputs the binary representation of the integer using ℓ octets if ℓ is sufficiently large, or fails otherwise. $\mathsf{OS2IP}$ reverses this process: given a string, it interprets it as the binary representation of an integer and outputs that integer.

Parameters. The scheme is parameterized by k, which is the bit length of the RSA modulus (strictly speaking, there are two parameters: *kLen* and *kBits*, representing its length in bytes and in bits, respectively; but for the purposes of the analysis the bit length is sufficient). The value $\mathit{emLen} = \lceil (k-1)/8 \rceil$ denotes the number of octets needed to represent a PSS encoding; i.e., a PSS encoding will always take up exactly $k-1$ bits.

As in PSS, the choice of the functions Hash and MGF and the parameters *hLen* and *sLen* are additional design choices (parameters, if you will) that define an instantiation of the scheme. The value *hLen* denotes the length in octets of the output of the hash function Hash that's used in the scheme. It is important that *hLen* be set up in such a way that, in the random oracle model, the probability that two distinct inputs to $\mathsf{Hash}(\cdot, \mathit{hLen})$ yield the same output (i.e. collide) be minuscule; an adversary whose running time is t can generate at most t such inputs; thus $2^{4\mathit{hLen}}$ needs to be a generous upper bound on t. The value *sLen* denotes the length (in octets) of the salt of the PSS encoding.

Our security analysis requires that $\mathit{emLen} \geq \max(2\mathit{hLen}, \mathit{hLen} + \mathit{sLen}) + 2$.

PSS encoding and decoding procedures. Recall that, in RSA-PSS, the signing algorithm is broken down into two steps. The first step does not involve the secret key: it simply encodes the input message in a special way. The second step uses the secret key in order to compute the signature corresponding to the encoding obtained in step one. Analogously, signature verification consists of two steps as well: the first step uses the public key in order to compute what may turn out to be an encoding of the message; the second step verifies that the string obtained in step one is indeed a valid encoding of the message.

When describing RSA-BSSA below, we invoke the encoding and decoding procedures from the IETF standard [32]: $\mathsf{PSSEncode}(\mathtt{msg}, \ell)$ is the function that, on input a message $\mathtt{msg}$ and an integer ℓ, produces a string $\mathtt{EM}$ (encoded message) of $\lceil \ell/8 \rceil$ octets whose ℓ rightmost bits constitute a PSS encoding of $\mathtt{msg}$. $\mathsf{PSSVerify}(\mathtt{msg}, \mathtt{EM}, \ell)$ verifies that $\mathtt{EM}$ is consistent with the output of $\mathsf{PSSEncode}(\mathtt{msg}, \ell)$. For an RSA modulus of bit length k, the PSS scheme will use $\ell = k - 1$, so $\mathtt{EM}$ will be of length $emLen = \lceil (k-1)/8 \rceil$.

Specifically (but briefly), $\mathsf{PSSEncode}(\mathtt{msg}, \ell)$ works as follows: first, hash $\mathtt{msg}$ to obtain $\mathtt{mHash} = \mathsf{Hash}(\mathtt{msg}, hLen)$, and pick a random string $\mathtt{salt}$ of length $sLen$ bytes (octets). Compute $H = \mathsf{Hash}(0^{64} \circ \mathtt{mHash} \circ \mathtt{salt})$, and use it to compute a mask $\mathtt{dbMask} = \mathsf{MGF}(H, emLen - hLen - 1)$ and use it to mask the salt: $\mathtt{maskedDB} = \mathtt{DB} \oplus \mathtt{dbMask}$, where $\mathtt{DB}$ is $\mathtt{salt}$ padded (to make sure that the resulting string is of the correct length) with a pre-defined string. Then output the encoded message $\mathtt{EM} = \mathtt{maskedDB} \circ H \circ 0xBC$.

In turn, $\mathsf{PSSVerify}(\mathtt{msg}, \mathtt{EM}, \ell)$ begins by parsing $\mathtt{EM} = \mathtt{maskedDB} \circ H \circ 0xBC$. Then it computes $\mathtt{dbMask}$ as above to unmask $\mathtt{salt}$ from $\mathtt{maskedDB}$ (it rejects if the padding was incorrect) and verifies that $H = \mathsf{Hash}(0^{64} \circ \mathtt{mHash} \circ \mathtt{salt})$ for $\mathtt{mHash} = \mathsf{Hash}(\mathtt{msg}, hLen)$. In Appendix B, we provide more details about the history of PSS. In Appendix C we provide a more detailed description of the verification algorithm for PSS.

3.1 The basic scheme

We begin by describing the basic scheme in which the user obtains from the signer an RSA-PSS signature on the message $\mathtt{msg}$. As usual, the scheme consists of a key generation algorithm, a protocol for obtaining signatures, and a signature verification algorithm.

Key generation The key generation algorithm is the standard RSA key generation: The public key is $PK = (N, e)$, where N is an RSA modulus of length k (k is given as input to the key generation algorithm, in unary, i.e. 1^k), and e is relatively prime to $\varphi(N)$. The secret key is $SK = (N, d)$ such that $ed \equiv 1 \bmod \varphi(N)$. The exact specification is as described in the PKCS#1 standard.

Blind signing protocol The protocol consists of three algorithms: Blind, BSig, and $\mathsf{Finalize}$. On input a message $\mathtt{msg}$ that the client wishes to get a signature for, the client runs $\mathsf{Blind}(PK, \mathtt{msg})$ and obtains $(\mathtt{blinded\_msg}, \mathtt{inv})$. The server runs the algorithm $\mathsf{BSig}(SK, \mathtt{blinded\_msg})$ and outputs a blinded signature $\mathtt{blinded\_sig}$. The user runs $\mathsf{Finalize}(PK, \mathtt{msg}, \mathtt{blinded\_sig}, \mathtt{inv})$ to derive the signature σ. The three algorithms are as follows:

$\mathsf{Blind}(PK, \mathtt{msg})$ Compute a PSS encoding of $\mathtt{msg}$: $\mathtt{EM} = \mathsf{PSSEncode}(\mathtt{msg}, k - 1)$, and let $m = \mathsf{OS2IP}(\mathtt{EM})$ be the corresponding integer. Next, sample

$r \leftarrow \mathbb{Z}_N^*$, compute $z = mr^e \bmod N$, and make sure that $z \in \mathbb{Z}_N^*$. Compute $r_{inv} = r^{-1} \bmod N$, and output (z, r_{inv}) as octet strings, i.e., output $\texttt{blinded\_msg} = \mathsf{I2OSP}(z, kLen)$, $\texttt{inv} = \mathsf{I2OSP}(r_{inv}, kLen)$.

$\mathsf{BSig}(SK, \texttt{blinded\_msg})$ First, check that the string `blinded_msg` is of bit length k, and reject if it is not. Next, convert into a k-bit integer $m = \mathsf{OS2IP}(\texttt{blinded\_msg})$. Output the binary representation of $s = m^d \bmod N$, i.e. $\texttt{blinded\_sig} = \mathsf{I2OSP}(s, kLen)$.

$\mathsf{Finalize}(PK, \texttt{msg}, \texttt{blinded\_sig}, \texttt{inv})$ Convert `blinded_sig` and `inv` into integers using the OS2IP procedure: $z = \mathsf{OS2IP}(\texttt{blinded\_sig})$, $r_{inv} = \mathsf{OS2IP}(\texttt{inv})$; compute $s = zr_{inv} \bmod N$. The signature is the binary representation of s, i.e. $\sigma = \mathsf{I2OSP}(s, kLen)$. Finally, if $\mathsf{PSSVerify}(PK, \texttt{msg}, \sigma)$, then output σ, else fail.

Verification The verification algorithm calls $\mathsf{PSSVerify}(PK, \texttt{msg}, \sigma)$. (Described in more detail in Appendix C.)

The following theorem follows easily by inspection:

Theorem 1. *The RSA-BSSA scheme is correct.*

As we will show in Section 5, it also satisfies one-more unforgeability under the one-more-RSA assumption [6]. However, as we will explain in more detail in Section 4, it is not clear whether or not this construction satisfies blindness.

3.2 RSA-BSSA, version A

The basic construction described above (Section 3.1) results in a perfectly blind signing protocol whenever $PK = (N, e)$ where the public exponent e is relatively prime to $\varphi(N)$: in that case, for any $m \in \mathbb{Z}_N^*$, selecting $r \leftarrow \mathbb{Z}_N^*$ uniformly at random and outputting $z = mr^e$ ensures that z is a uniformly random element of $\mathbb{Z}_N^*$. This is because, of course, if there exists d such that $ed \equiv 1 \bmod \varphi(N)$, then for each $z \in \mathbb{Z}_N^*$, there exists a unique $r = (z/m)^d$ such that $z = mr^e$.

Thus, in order to ensure blindness, it is sufficient to ensure that e is relatively prime to $\varphi(N)$. Consider the following variant of RSA-BSSA, in which a public key contains a proof that e is relatively prime to $\varphi(N)$ as described by Goldberg, Reyzin, Sagga and Baldimtsi [22].

It will require the additional parameter κ, which is a statistical security parameter. Further, it will require a function R_k that, on input three integers, outputs a random integer $0 \leq a < 2^{k-1}$; such R_k can be constructed, for example, from MGF. Let e' be a prime that is small enough that checking that $(N, e) \in L_{e'}$ can be done efficiently, where $L_{e'} = \{(N, e) \mid N, e > 0 \text{ and no prime less than } e' \text{ divides } e\}$. In practice, e is often prime and small enough that setting $e' = e$ works. We let e' be a system-wide parameter, so each procedure below receives it as input. Finally, let $\ell = \lceil \kappa / \log_2(e') \rceil$.

Key generation On input the desired modulus length k and a statistical security parameter κ, run the RSA key generation algorithm as in the basic

protocol (Section 3.1) to obtain (N, e) and d. Next, compute a proof π that e is relatively prime to $\varphi(N)$, as follows: for $1 \leq i \leq \ell$, let $a_i = R_k(N, e, i)$, compute $b_i = {a_i}^d \bmod N$, and let $\pi = b_1, \ldots, b_\ell$.
The public key is $PK = (N, e, \pi)$, the secret key is $SK = (N, d)$.

Blind signing protocol Before running the signing protocol, the user verifies that the public key $PK = (N, e, \pi)$ is well-formed: let $\pi = b_1, \ldots, b_\ell$; for $1 \leq i \leq \ell$, check that $b_i^e = R_k(N, e, i) \bmod N$. If one of the checks fails, fail. Else, run the Blind, BSig, and $\mathsf{Finalize}$ algorithms as described in Section 3.1.

Verification As in Section 3.1, return $\mathsf{PSSVerify}(PK, \mathtt{msg}, \sigma)$.

As we will see in Section 4.3, using the Goldberg et al. [22] proof system ensures that blindness holds.

3.3 RSA-BSSA, version B

As we will see in Section 4.3, another way to ensure blindness is to modify the construction in such a way that the value `mHash` incorporated into the PSS encoding of the message to be signed reveals nothing about this message. This calls for a simple modification of the basic protocol that requires that, instead of invoking the signing protocol directly on the message `msg`, the user invokes it on the message $\mathtt{msg}' = \mathtt{msg} \circ \mathtt{rand}$, where `rand` is a random value of κ bits, where κ is a security parameter. More precisely:

Key generation Run the RSA key generation algorithm as in the basic protocol (Section 3.1) to obtain $PK = (N, e)$ and $SK = d$.

Blind signing protocol The user generates a random string `rand` of κ bits, and runs the signing protocol in Section 3.1 on input $\mathtt{msg}' = \mathtt{msg} \circ \mathtt{rand}$, and obtains from it the signature σ' on the message $\mathtt{msg}'$. Output the signature $\sigma = (\sigma', \mathtt{rand})$.

Verification Following Section 3.1, on input `msg` and $\sigma = (\sigma', \mathtt{rand})$, the verification algorithm makes sure that `rand` consists of κ bits, rejects if it does not, and then returns the output of $\mathsf{PSSVerify}(PK, \mathtt{msg}', \sigma')$, where $\mathtt{msg}' = \mathtt{msg} \circ \mathtt{rand}$.

4 Blindness of RSA-BSSA

4.1 Preliminaries

In the blindness experiment, the adversary picks the modulus N; thus we cannot assume that it is a proper RSA modulus. Therefore, in order to understand how much information such an adversary can learn in the blindness experiment, we must consider the structure of the group $\mathbb{Z}_N^*$ for arbitrary N.

Lemma 1. *Let $N > 1$ be any odd integer, let $N = \prod_{i=1}^{\ell} p_i^{\alpha_i}$ be is its prime factorization. Then $\mathbb{Z}_N^*$ is of size $\varphi(N) = \prod_{i=1}^{\ell} \varphi(p_i^{\alpha_i}) = \prod_{i=1}^{\ell} p_i^{\alpha_i - 1}(p_i - 1)$ is isomorphic to $\mathbb{Z}_{\varphi(p_1^{\alpha_1})} \times \mathbb{Z}_{\varphi(p_2^{\alpha_2})} \times \ldots \times \mathbb{Z}_{\varphi(p_\ell^{\alpha_\ell})}$.*

For the proof, we refer to Section 7.5 of Shoup [36]. The lemma implies that every element $x \in \mathbb{Z}_N^*$ can be viewed as a vector $(x_1, \ldots, x_\ell) \in \mathbb{Z}_{p_1^{\alpha_1-1}(p_1-1)} \times \mathbb{Z}_{p_2^{\alpha_2-1}(p_2-1)} \times \ldots \times \mathbb{Z}_{p_\ell^{\alpha_\ell-1}(p_\ell-1)}$, and vice versa.

Let Ψ_N denote this isomorphism; when N is clear from the context, we will write it as Ψ. Moreover, there is a (not necessarily efficient) algorithm that computes Ψ_N, as follows: on input $x \in \mathbb{Z}_N^*$, compute $x_i = x \bmod p_i^{\alpha_i}$, and then find $\chi_i \in \mathbb{Z}_{p_i^{\alpha_i-1}(p_i-1)}$ such that $x = g_i^{\chi_i}$, where g_i is a generator of $\mathbb{Z}_{p_i^{\alpha_i}}^*$ (it exists by Theorem 7.28 in Shoup [36]).

We also refer the reader to Shoup [36] for the Chinese Remainder Theorem; below, by $CRT(x_1, \ldots, x_\ell)$ we denote the element $x \in \mathbb{Z}_N$ such that $x = x_i \bmod p_i^{\alpha_i}$, where $N = \prod_{i=1}^{\ell} p_i^{\alpha_i}$ is the prime factorization of N.

Definition 7 (Roots and residues). *Let N and e be any positive integers, and let $m \in \mathbb{Z}_N^*$. Let the set $Roots_{N,e}(m) = \{s \in \mathbb{Z}_N^* \mid s^e = m\}$. Let the set $Residues_{N,e} = \{m \in \mathbb{Z}_N^* \mid Roots_{N,e}(m) \neq \emptyset\}$.*

Lemma 2. *Let $N > 1$ be any odd integer. If an integer e is relatively prime to $\varphi(N)$, then the distribution $D_0(N, e) = \{r \leftarrow \mathbb{Z}_N^* \ : \ r\}$ is identical to the distribution $D_1(N, e) = \{r \leftarrow \mathbb{Z}_N^* \ : \ r^e\}$.*

This lemma is well-known, but we include a proof for completeness and also to see what goes wrong if e is not relatively prime to $\varphi(N) = \prod_{i=1}^{\ell} p_i^{\alpha_i-1}(p_i-1)$.

Proof. By Lemma 1, $r \in \mathbb{Z}_N^*$ corresponds (via the isomorphism Ψ in Lemma 1) to a unique $(r_1, \ldots, r_\ell) \in \mathbb{Z}_{p_1^{\alpha_1-1}(p_1-1)} \times \mathbb{Z}_{p_2^{\alpha_2-1}(p_2-1)} \times \ldots \times \mathbb{Z}_{p_\ell^{\alpha_\ell-1}(p_\ell-1)}$; similarly, $q = r^e$ corresponds to $(q_1, \ldots, q_\ell) = (er_1, \ldots, er_\ell)$. Moreover, since e is relatively prime to $p_i^{\alpha_i-1}(p_i-1)$, by the extended Euclidean algorithm, there exists $d_i \in \mathbb{Z}_{p_i^{\alpha_i-1}(p_i-1)}$ such that $ed_i = 1 \bmod p_i^{\alpha_i-1}(p_i-1)$. Therefore, sampling r at random from $\mathbb{Z}_N^*$ results in the same distribution as choosing $q \leftarrow \mathbb{Z}_N^*$ and then letting r be the value that corresponds (via the isomorphism Ψ^{-1}) to $(r_1, \ldots, r_\ell) = (d_1q_1, \ldots, d_\ell q_\ell)$. Thus, the outputs of D_0 and D_1, r and $q = r^e$, are distributed identically. □

Lemma 3. *Let $p > 2$ be a prime number, and let $e \geq 2$ and $\alpha \geq 1$ be integers. Let $g = \gcd(e, p^{\alpha-1}(p-1))$. Then for any $m \in Residues_{p^\alpha,e}$, $|Roots_{p^\alpha,e}(m)| = g$. I.e. either $m \notin Residues_{p^\alpha,e}$), or it has exactly g e^{th} roots.*

Proof. First, let us show that, if $m \in Residues_{p^\alpha,e}$, then $|Roots_{p^\alpha,e}(m)| \geq g$. Let us write $e = gf$ and $\varphi(p^\alpha) = p^{\alpha-1}(p-1) = gq$. Let s be such that $s^e = m$; we know that s exists since $m \in Residues_{p^\alpha,e}$. Let $\sigma = \Psi_{p^\alpha}(s)$, $\mu = \Psi(m)$. Consider the set $S = \{s_k \mid 0 \leq k \leq g-1, s_k = \Psi^{-1}(\sigma + kq)\}$. Note that $|S| = g$, since $kg = k'g \Rightarrow k = k' \bmod \varphi(p^\alpha) = gq$. Also note that for any $s_k \in S$, $\Psi(s_k^e) = e(\sigma + iq) = e\sigma + keq = \mu + ifgq = \mu + if\varphi(p^\alpha) = \mu$; thus, $s_k^e = m$. Therefore, $S \subseteq Roots_{p^\alpha,e}(m)$, and thus $|Roots_{p^\alpha,e}(m)| \geq g$.

Next, let us show the other direction. Since g is the greatest common divisor of e and $\varphi(p^\alpha)$, there exists $1 \leq h \leq \varphi(p^\alpha)$ such that $fh = 1 \bmod \varphi(p^\alpha)$. Observe that $|Residues_{p^\alpha,e}| \geq \varphi(p^\alpha)/g = q$. To see that this is the case, note that for $0 \leq u < q$, $m = \Psi^{-1}(gu) \in Residues_{p^\alpha,e}$, since $(\Psi^{-1}(hu))^e = \Psi^{-1}(ehu) = \Psi^{-1}(gfhu) = \Psi^{-1}(gu) = m$. Since there are at least q residues, and (as we just saw) each has at least g e^{th} roots, no element of $Residues_{p^\alpha,e}$ can have more than $\varphi(p^\alpha)/q = g$ e^{th} roots. □

As part of the proof of Lemma 3, we also proved the following:

Corollary 1. *Let $p > 2$ be a prime number, and let $e > 1$ and $\alpha \geq 1$ be integers. Let $g = \gcd(e, p^{\alpha-1}(p-1))$, and let $q = p^{\alpha-1}(p-1)/g$. Let $m \in Residues_{p^\alpha,e}$, and let $s \in Roots_{p^\alpha,e}(m)$. Then $Roots_{p^\alpha,e}(m) = \{s_k \mid 0 \leq k \leq g-1, s_k = \Psi^{-1}(\sigma + kq)\}$, where $\sigma = \Psi(s)$.*

Combining Lemma 3 and Corollary 1 with the Chinese Remainder Theorem, we get:

Lemma 4. *Let $N > 1$ be an odd integer, and let $\prod_{i=1}^{\ell} p_i^{\alpha_i}$ be its prime factorization. Let $e > 1$ be an integer. Let $g_i = \gcd(e, p_i^{\alpha_i - 1}(p_i - 1))$. Then for any $m \in Residues_{N,e}$, $|Roots_{N,e}(m)| = \prod_{i=1}^{\ell} g_i$.*

Corollary 2. *Let $N > 1$ be an odd integer, and let $\prod_{i=1}^{\ell} p_i^{\alpha_i}$ be its prime factorization. Let $e > 1$ be an integer. Let $g_i = \gcd(e, p_i^{\alpha_i-1}(p_i - 1))$, and let $q_i = p_i^{\alpha_i-1}(p_i-1)/g_i$. Let $m \in Residues_{N,e}$, let s be its e^{th} root, and let $\Psi(s) = (\sigma_1, \ldots, \sigma_\ell)$. Then $Roots_{N,e}(m) = \{CRT(s_{1,k_1}, \ldots, s_{\ell,l_\ell}) \mid \forall 1 \leq i \leq \ell, 0 \leq k_i \leq g_i - 1, s_{i,k_i} = \Psi^{-1}_{p_i^{\alpha_i}}(\sigma_i + k_i q_i)\}$.*

Lemma 5. *Let $N > 1$ be an odd integer, and $e > 1$ be an integer. Then r selected as follows is a uniformly random element of $\mathbb{Z}_N^*$: first, select y uniformly at random from $Residues_{N,e}$. Then, select r uniformly at random from $Roots_{N,e}(y)$.*

Proof. By Lemma 4, every element of $Residues_{N,e}$ has the same number of roots, and so selecting a random element of $Residues_{N,e}$ and then picking one of its roots at random is equivalent to picking a random element of $\mathbb{Z}_N^*$. □

Lemma 6. *Let $N > 1$ be an odd integer, and $e > 1$ be an integer. Let $m \in Residues_{N,e}$. Let z be selected uniformly at random from $Residues_{N,e}$; let $y = z/m$. Then y is a uniformly random element of $Residues_{N,e}$.*

Proof. Let $y \in Residues_{N,e}$. y is selected whenever the experiment chooses $z = my$; this happens with probability $1/|Residues_{N,e}|$. □

In our analysis below, it will be important that even if the adversary picks an e^{th} root u of the value $z = mr^e$ (recall that z is what the signature recipient sends to the signer in order to get the message signed), it still cannot alter the distribution of the resulting signature. We will see that the signature $s = u/r$ is a member of $Roots_{N,e}(m)$ that is independent of u as long as r had been picked uniformly at random. In other words, as long as r is picked uniformly at random, s is random as well, no matter what the adversary does. This is captured in the following lemma:

Lemma 7. *Let $N > 1$ be an odd integer, and $e > 1$ be an integer. Then for all $m \in Residues_{N,e}$, $z \in Residues_{N,e}$, $u \in Roots_{N,e}(z)$, the following outputs a uniformly random element of $Roots_{N,e}(m)$: pick $r \leftarrow Roots_{N,e}(z/m)$, output u/r.*

Proof. Consider $N = p^\alpha$ for some prime p; the general case follows via the Chinese Remainder Theorem. Let s_0 be the smallest (in absolute value) element of $Roots_{p^\alpha,e}(m)$, and let r_0 be the smallest element of $Roots_{p^\alpha,e}(z/m)$. Let $\Psi(s_0) = \sigma$, $\Psi(r_0) = \rho$, $\Psi(u) = \upsilon$, $g = \gcd(e, p^{\alpha-1}(p-1))$, and $q = p^{\alpha-1}(p-1)/g$.

By Corollary 1, $Roots_{p^\alpha,e}(m) = \{s_k \mid 0 \leq k \leq g-1, s_k = \Psi^{-1}(\sigma + kq)\}$. Since u/r_0 is an e^{th} root of m, $u/r_0 = \Psi^{-1}(\sigma + nq)$ for some $0 \leq n < g$. Also by Corollary 1, $Roots_{p^\alpha,e}(m/z) = \{r_k \mid 0 \leq k \leq g-1, r_k = \Psi^{-1}(\rho+kq)\}$. Selecting r_k uniformly at random corresponds to picking $k \leftarrow \{0, \ldots, g-1\}$, and results in outputting $u/r_k = \Psi^{-1}(\upsilon-(\rho+kq)) = \Psi^{-1}((\upsilon-\rho)-kq) = \Psi^{-1}((\sigma+nq)-kq) = \Psi^{-1}(\sigma + (n-k)q)$. Since k is random, $n - k \bmod g$ is also a random element of $\{0, \ldots, g-1\}$, and therefore the output $r_k u = s_{(n-k) \bmod g}$ is uniformly random element of $Roots_{p^\alpha,e}(m)$. □

4.2 Blindness of the signing protocol

Let us consider $\mathcal{A}$'s interaction with the blindness challenger, and then analyze what information $\mathcal{A}$ learns as a result of this interaction. For simplicity, below we omit the integer-to-string conversions and, when clear from context that integers in question are elements of $\mathbb{Z}_N^*$, we omit "mod$N$."

$\mathcal{A}$ is invoked $\mathcal{A}(1^k)$ selects a public key $PK = (N, e)$ and two messages $\texttt{msg}_0$ and $\texttt{msg}_1$.

$\mathcal{A}$ acts as the blind signer For $j \in \{0, 1\}$, the challenger computes $\texttt{EM}_j = \mathsf{PSSEncode}(\texttt{msg}_j, k-1)$; let $m_j = \mathsf{OS2IP}(\texttt{EM}_j)$ be the corresponding integer. Next, sample $r_j \leftarrow \mathbb{Z}_N$, compute $z_j = m_j r_j^e$. Compute $\texttt{inv}_j = r_j^{-1}$. The challenger sends to $\mathcal{A}$ the values z_b and z_{1-b}.

$\mathcal{A}$ receives the signatures Upon receipt of u_b and u_{1-b} from the signer, the challenger computes $s_0 = u_0/r_0$ and $s_1 = u_1/r_1$. If both signatures verify, i.e. $s_0^e = m_0$ and $s_1^e = m_1$, it sends (s_0, s_1) to $\mathcal{A}$; else it sends $\perp$ to $\mathcal{A}$.

$\mathcal{A}$'s output $\mathcal{A}$ outputs some value $\texttt{output}$.

Claim 1 *If e is relatively prime to $\varphi(N)$, then z_0 and z_1 (sent to $\mathcal{A}$ while it is acting as the blind signer) are both random elements of $\mathbb{Z}_N^*$ and are distributed independently of b, m_0 and m_1.*

Proof. Follows immediately from Lemma 2. □

Claim 2 *If e is relatively prime to $\varphi(N)$, then $\mathcal{A}$'s view after receiving the signatures is independent of the bit b.*

Proof. $\mathcal{A}$ already knows, based on the values u_b and u_{1-b} it sent to the challenger in the previous step, whether it will receive the signatures or $\perp$. If it receives the signatures, then there are unique values $r_{0,b}, r_{1,b}$ consistent with either $b \in \{0,1\}$, and they were equally likely to have been chosen; see Lemma 2. If $\mathcal{A}$ does not receive the signatures, then $\mathcal{A}$ learns nothing. □

If e is not relatively prime to $\varphi(N)$, then there are two cases, based on whether the signatures output by Finalize pass verification. The easy case is when the signatures output by Finalize do not both pass verification; then, the challenger sends $\perp$ to the adversary and thus no additional information is revealed in this step. Let us show that:

Claim 3 *If both signatures verify, $s_0^e = m_0$ and $s_1^e = m_1$, then $\mathcal{A}$'s view in the blindness experiment is independent of b.*

Proof. Let us condition on the event that the signatures pass verification. In this case, the values m_0, m_1 computed by the challenger, as well as the value z_b and z_{1-b} the challenger sent to the signer must all be in the set $\mathit{Residues}_{N,e}$. Let us consider a series of experiments.

Our first experiment is the case of running the blindness challenger with $b = 0$: The challenger begins by sampling m_0 and m_1 as PSS encodings of $\mathtt{msg}_0$ and $\mathtt{msg}_1$. Then, it samples $r_0 \leftarrow \mathbb{Z}_N^*$, $r_1 \leftarrow \mathbb{Z}_N^*$, computes $z_0 = m_0 r_0^e$ and $z_1 = m_1 r_1^e$, and sends $(z, z') = (z_0, z_1)$ to the adversary. The adversary responds with (u_0, u_1), and the challenger computes the signatures $s_0 = u_0/r_0$ and $s_1 = u_1/r_1$.

By Lemma 5, instead of choosing r_0 and r_1 uniformly at random from $\mathbb{Z}_N^*$ and then setting $z_0 = r_0^e m_0$ and $z_1 = r_1^e m_1$, one could equivalently choose y_0, y_1 uniformly at random from $\mathit{Residues}_{N,e}$, and then let $z_0 = y_0 m_0$, $z_1 = y_1 m_1$, $r_0 \leftarrow \mathit{Roots}_{N,e}(y_0)$, $r_1 \leftarrow \mathit{Roots}_{N,e}(y_1)$; let us call the resulting experiment A_0. By Lemma 6, this is equivalent to choosing z_0 and z_1 uniformly at random from $\mathit{Residues}_{N,e}$, and letting $r_0 \leftarrow \mathit{Roots}_{N,e}(z_0/m_0)$, $r_1 \leftarrow \mathit{Roots}_{N,e}(z_1/m_1)$; let us call the resulting experiment B_0. By Lemma 7, this is equivalent to picking z_0 and z_1 uniformly at random from $\mathit{Residues}_{N,e}$ and sending the adversary the pair $(z, z') = (z_0, z_1)$, and upon receipt of u_0 and u_1 such that $u_0^e = z_0$ and $u_1^e = z_1$, outputting $s_0 \leftarrow \mathit{Roots}_{N,e}(m_0)$, $s_1 \leftarrow \mathit{Roots}_{N,e}(m_1)$; let us call the resulting experiment C_0.

Let us obtain a new experiment, C_1, by modifying C_0: let $(z, z') = (z_1, z_0)$, while everything else stays the same. C_1 gives the adversary identical view to C_0. Let B_1 be the same as B_0 except for $(z, z') = (z_1, z_0)$; by Lemma 7, the adversary's view here is identical to C_1. Let A_1 be identical to A_0 except $(z, z') = (z_1, z_0)$; by Lemma 6, it is identical to B_1. Finally, by Lemma 5, A_1 gives the adversary the same view as the challenger when $b = 1$. □

Rephrasing Claims 1, 2 and 3, we get the following two lemmas:

Lemma 8. *Let $E^{\mathcal{A}}_{relprime}$ be the event that $\mathcal{A}$ playing the blindness game with the challenger for the basic version of RSA-BSSA sets $PK = (N, e)$ such that e is relatively prime to $\varphi(N)$. Conditioned on $E^{\mathcal{A}}_{relprime}$, $\mathcal{A}$ receives the same view in the blindness experiment for $b = 0$ as for $b = 1$.*

Lemma 9. *Let $E^{\mathcal{A}}_{goodsigs}$ be the event that in the blindness game with adversary $\mathcal{A}$, the challenger for the basic version of RSA-BSSA obtains two signatures that pass verification. Conditioned on $E^{\mathcal{A}}_{goodsigs}$, $\mathcal{A}$ receives the same view in the blindness experiment for $b = 0$ as for $b = 1$.*

When blindness might not hold. Based on the above analysis, the only situation in which $\mathcal{A}$'s view may depend on b is when e is not relatively prime to $\varphi(N)$ and the challenger fails to output two valid signatures. In this situation, z_b may leak enough information about m_b that it might be possible to infer whether $m_b = \mathsf{PSSEncode}(\mathtt{msg}_0)$ or $m_b = \mathsf{PSSEncode}(\mathtt{msg}_1)$, revealing b.

For example, for a prime p such that $e \mid p - 1$, x and y are in the same e^{th} residue class modulo p if there exists $r \in \mathbb{Z}_p^*$ such that $x = yr^e \bmod p$. There are e distinct e^{th} residue classes modulo p when $e \mid p - 1$; they correspond to the e values of $\Psi_p(x) \bmod e$. Thus, determining e^{th} residue class modulo p of an unknown x provides $\log e$ bits of information about x.

Suppose $N = \prod_{i=1}^{\ell} p_i$ such that $e \mid p_i - 1$ for $1 \leq i \leq \ell$, where each p_i is a distinct prime number. Then $z_b = m_b r^e \bmod p_i$ is the same e^{th} residue class as $m_b \bmod p_i$. Thus, z_b reveals $\ell \log e$ bits of information about m_b. If each p_i is only slightly larger than e, then z reveals (in the information-theoretic sense) more than half the bits of m_b. It is unclear how these information-theoretic bits correspond to physical bits; therefore, we must consider the worst case, in which they reveal a significant number of bits of the encoded message `EM`. Especially devastating would be the case when the revealed bits correspond to H and the bits of `maskedDB` just to the left of H; by XORing those bits with $\mathsf{MGF}(H, lenDB)$, $\mathcal{A}$ can recover `salt`, and check whether $H = \mathsf{Hash}(M')$ where M' encodes $\mathtt{mHash} = \mathsf{Hash}(\mathtt{msg}_0, hLen)$ with `salt` (which corresponds to $b = 0$) or $\mathtt{mHash} = \mathsf{Hash}(\mathtt{msg}_1, hLen)$ with `salt` (which corresponds to $b = 1$).

As we will see below, variants A and B of RSA-BSSA prevent this situation in two distinct ways. Variant A makes it extremely unlikely that e is not relatively prime to $\varphi(N)$. Variant B ensures that recovering `mHash` does not help in checking whether it corresponds to $\mathtt{msg}_0$ or $\mathtt{msg}_1$: any value `mHash` is equally likely to correspond to either, depending on the choice of the randomizer `rand`.

4.3 Blindness of variants A and B

Theorem 2. *RSA-BSSA, Version A, satisfies blindness (Definition 5).*

Proof. This follows by the soundness of the proof due to Goldberg et al. [22]. □

Theorem 3. *RSA-BSSA, Version B, satisfies blindness in the random-oracle model (Definition 5).*

Proof. For $j \in \{0, 1\}$, let m_j be the integer that corresponds to $\mathsf{PSSEncode}(\mathtt{msg}_j \circ \mathtt{rand}_j)$ for a random string $\mathtt{rand}_j$ of κ bits, and $z_j = m_j r_j^e$. Let $(z, z') = (z_b, z_{1-b})$ be the values that the challenger sends to the adversary $\mathcal{A}$ in the blindness experiment with the bit b. In order to see that (z, z') are distributed independently of the bit b it is sufficient to show that $\mathtt{mHash}_b = \mathsf{Hash}(\mathtt{msg}_b \circ \mathtt{rand}_b, hLen)$ is distributed independently of b for a randomly chosen $\texttt{rand}_b$, since $\mathsf{PSSEncode}$ just feeds its input string to Hash.

Let us model Hash as a random oracle. Consider a modified blindness experiment in which the challenger also controls the random oracle Hash:

$\mathcal{A}$ is invoked $\mathcal{A}(1^k)$ selects a public key $PK = (N, e)$ and $\texttt{msg}_0$ and $\texttt{msg}_1$.

$\mathcal{A}$ acts as the blind signer For $j \in \{0, 1\}$, compute $\texttt{EM}_j = \mathsf{PSSEncode}(\texttt{msg}_j, k-1)$ differently from the blindness challenger, as follows: instead of picking $\texttt{rand}_j$ first, and then setting $\texttt{mHash}_j$, leave $\texttt{rand}_j$ undefined for now and let $\texttt{mHash}_j$ be a random string of length $8hLen$. Next, follow the protocol and let $m_j = \mathsf{OS2IP}(\texttt{EM}_j)$ be the corresponding integer. Next, sample $r_j \leftarrow \mathbb{Z}_N^*$, compute $z_j = m_j r_j^e \bmod N$ and make sure $z_j \in \mathbb{Z}_N^*$. Compute $\texttt{inv}_j = r_j^{-1} \bmod N$. The challenger sends to $\mathcal{A}$ the values z_b and z_{1-b}.

$\mathcal{A}$ receives the signatures Upon receipt of u_b and u_{1-b} from the signer, the challenger checks that $z_j = u_j^e$ for each $j \in \{0, 1\}$; if these fail, send $\perp$ to $\mathcal{A}$. If these checks pass, then choose random κ-bit strings $\texttt{rand}_0$ and $\texttt{rand}_1$ and set the random oracle so that $\texttt{mHash}_j = \mathsf{Hash}(\texttt{msg}_j \circ \texttt{rand}_j, hLen)$; if setting the random oracle this way fails (i.e., the value $\mathsf{Hash}(\texttt{msg}_j \circ \texttt{rand}_j, hLen)$ is already defined), then this experiment fails.
Else, for each $j \in \{0, 1\}$, compute $s_j = u_j / r_j$ and send (s_0, s_1) to $\mathcal{A}$.

$\mathcal{A}$ queries Hash Since in this modified blindness experiment, the challenger controls the random oracle, we must also describe how it handles the adversary's queries to Hash. As usual, when $\mathcal{A}$ queries a value (v, ℓ) such that $\mathsf{Hash}(v, \ell)$ has not yet been defined, respond with a random string of length 8ℓ; when querying for a string whose value has already been defined, return that value.

$\mathcal{A}$'s output At the end of its execution, $\mathcal{A}$ produces some output. At that point, if $\texttt{rand}_0$ and $\texttt{rand}_1$ are still undefined, choose random κ-bit strings $\texttt{rand}_0$ and $\texttt{rand}_1$ and set the random oracle so that $\texttt{mHash}_j = \mathsf{Hash}(\texttt{msg}_j \circ \texttt{rand}_j, hLen)$; if setting the random oracle this way fails (i.e., the value $\mathsf{Hash}(\texttt{msg}_j \circ \texttt{rand}_j, hLen)$ is already defined), then this experiment fails.

Our theorem will follow by putting together the following three claims:

Claim 4 *Conditioned on the event that the modified blindness experiment does not fail, the view $\mathcal{A}$ receives in the modified blindness experiment above is independent of the bit b.*

Claim 5 *Conditioned on the event that the modified blindness experiment does not fail, the view $\mathcal{A}$ receives in the modified blindness experiment above is identical to the view it receives in the actual blindness experiment.*

Claim 6 *Let $\mathcal{A}$'s running time be t. Then the modified blindness experiment fails with probability $O(t2^{-\kappa})$.*

To see that the theorem follows from the claims, consider a sequence of experiments: (1) blindness game with $b = 0$; (2) modified blindness game with $b = 0$; (3) modified blindness game with $b = 1$; (4) blindness game with $b = 1$.

(1) and (2) are indistinguishable by combining Claims 5 and 6; similarly (3) and (4). (2) and (3) are indistinguishable by combining Claims 4 and 6.

We conclude our proof of the theorem by proving these claims.

Proof of Claim 4. This claim follows by construction. Note that in the step when $\mathcal{A}$ acts as the blind signer, the challenger does not even need to have b already defined: it can set $\texttt{mHash}_b$ and $\texttt{mHash}_{1-b}$ without knowing b and compute m_b and m_{1-b} from them; similarly it can sample r_b and compute z_b. If it needs to know b in the step where $\mathcal{A}$ receives the signatures, the challenger is already assured that m_0 and m_1 are both in $Residues_{N,e}$, and so by Lemma 9, $\mathcal{A}$'s view is independent of b.

Proof of Claim 5. In the random-oracle model, the only difference between the modified experiment above and the real blindness experiment is the point in time in which the values $\texttt{rand}_0$ and $\texttt{rand}_1$ are defined: whether they are already defined when $\mathcal{A}$ acts as the blind signer, or whether this does not happen until the step where $\mathcal{A}$ receives the signatures or (in the event it does not) the output step. If we condition on the event that the modified experiment does not fail, then we know that $\mathcal{A}$ has never over the course of its execution queried Hash on the values $(\texttt{msg}_0 \circ \texttt{rand}_0, hLen)$ and $(\texttt{msg}_1 \circ \texttt{rand}_1, hLen)$. In that case, whether $\texttt{rand}_0$ and $\texttt{rand}_1$ were already defined or not, is independent of $\mathcal{A}$'s view, and therefore the modified blindness experiment is identical to the original one.

Proof of Claim 6. The modified experiment fails if the adversary ever queries Hash on input $(\texttt{msg}_j \circ \texttt{rand}_j, hLen)$ for $j \in \{0,1\}$. In t steps, $\mathcal{A}$ may query at most t such strings. $\texttt{rand}_j$ is a random κ-bit string, so the probability it's among the t that $\mathcal{A}$ has queried, is $t2^{-\kappa}$. □

4.4 The basic version is a blind token scheme

Theorem 4. *The basic version of RSA-BSSA is a strongly unforgeable blind token scheme (Definition 6) under the one-more-RSA assumption.*

Proof. First, note that the basic version of RSA-BSSA satisfies the input-output specification for a two-move blind signature scheme (Definition 1) and the strong one-more unforgeability property (Definition 4). The first follows by inspection; the second, by Theorem 6. Thus, it is sufficient to show that for any $\mathcal{A}$, the view in the experiment described in Definition 6 is independent of the bit b.

Note that in the blind token security game, unless the challenger obtains two valid signatures, the adversary's view is independent of b based on how the game unfolds. Thus, an adversary $\mathcal{A}$ guessing b correctly in that game more often than half the time must be one for whom the challenger obtains two valid signatures. Then consider the following reduction $\mathcal{B}$ that plays the (usual) blindness game with a blind signature challenger for RSA-BSSA and uses $\mathcal{A}$ to contradict Lemma 9: it obtains from $\mathcal{A}$ the values (N, e) and the auxiliary data needed to sample the messages $\texttt{msg}_0$ and $\texttt{msg}_1$ and proceeds to sample them. Then it sends (N, e) and $\texttt{msg}_0$ and $\texttt{msg}_1$ to its challenger, and from then on, it passes messages back and forth from its challenger to $\mathcal{A}$, and outputs whatever $\mathcal{A}$ outputs. If $\mathcal{A}$ is successful, then $\mathcal{B}$ is successful. But $\mathcal{A}$ can only be successful (as we observed above) when the challenger outputs two valid signatures, and by Lemma 9, under these circumstances $\mathcal{B}$ cannot be successful, which is a contradiction. □

4.5 Blindness of Chaum-RSA-FDH

Consider Bellare et al. [6]'s version of Chaum blind signature; we will call it *Chaum-RSA-FDH* from now on. Chaum-RSA-FDH works as follows: Following RSA-FDH, the key generation algorithm generates an RSA key pair $PK = (N, e)$, $SK = d$, where $ed \equiv 1 \bmod \varphi(N)$. Following Chaum, in order to obtain a blind signature on a message M, the user first blinds it using a random $r \leftarrow \mathbb{Z}_N^*$ obtaining $z = \mathsf{Hash}(M) r^e \bmod N$. Then he sends z to the signer and gets back the blinded signature $y = z^d \bmod N$, and unblinds it to obtain and output $s = yr^{-1} \bmod N$. The resulting value passes RSA-FDH verification: $s^e = y^e r^{-e} = zr^{-e} = \mathsf{Hash}(M) r^e r^{-e} = \mathsf{Hash}(M)$.

Let (N, e) be such that e is not relatively prime to $\varphi(N)$. Let $U = \{u \mid u^e \equiv 1 \bmod N\}$; by Lemma 4, when e divides some prime factors of N, $|U| \geq e$. Let $\equiv_e$ be the following equivalence relation: $a \equiv_e b$ if there exist α, β and $u \in U$ such that $a = \alpha^e u \bmod N$ and $b = \beta^e u \bmod N$. It is easy to see that $\equiv_e$ partitions $\mathbb{Z}_N^*$ into $|U|$ equivalence classes. There is an efficient algorithm that, on input the factorization of N, a and b, determines whether $a \equiv_e b$. Moreover, for any $a, r \in \mathbb{Z}_N^*$, $a \equiv_e ar^e$.

In order to break blindness of Chaum-RSA-FDH, the adversary picks (N, e) such that it knows the factorization of N, and such that $e \mid \varphi(N)$. Next, it picks two messages M_0 and M_1 to send to the challenger, such that $1 \not\equiv_e \mathsf{Hash}(M_0) \not\equiv_e \mathsf{Hash}(M_1) \not\equiv_e 1$. The challenger computes $z_0 = \mathsf{Hash}(M_0) r_0^e$ and $z_1 = \mathsf{Hash}(M_1) r_1^e$, and sends them to the adversary in random order: (z_b, z_{1-b}). In order to determine the bit b, the adversary checks whether $z_b \equiv_e \mathsf{Hash}(M_0)$; if so, it returns 0, else it returns 1.

Bibliographic note. Before the community settled on what is now considered to be the right definition of blindness [1], the definition due to Juels, Ostrovsky and Luby [26] was the standard one. That definition's security experiment for blindness did not envision that the adversarial signer may generate the signing key in a malicious way, rather than following the key generation algorithm. Bellare, Namprempre, Pointcheval and Semanko showed that the Chaum-RSA-FDH scheme was a secure blind signature under the old definition [26]. As we saw, their result does not hold under the more modern definition of blindness that came several years after their paper came out. Fortunately, Chaum-RSA-FDH is a strongly one-more-unforgeable blind token scheme, i.e., it satisfies Definition 6.

Theorem 5. *The Chaum-RSA-FDH scheme described in Section 4.5 is a strongly one-more-unforgeable blind token scheme for any efficiently samplable message space $\mathcal{M}$.*

Proof. (Sketch) It is easy to see that the scheme satisfies the input-output structure and the correctness requirements. As for strong one-more unforgeability:

Bellare, Namprempre, Pointcheval and Semanko [6] showed that it was one-more-unforgeable under the one-more-RSA assumption. Strong one-more unforgeability follows because RSA-FDH is deterministic, i.e., there is a unique signature corresponding to each message. Thus we just need to show that for any $\mathcal{A}$, $\mathcal{A}$'s advantage in the blind token experiment described in Definition 6 is negligible; in fact we will see that it is 0.

Let $\mathcal{A}$ be an adversary playing the blind token game; let us consider the view $\mathcal{A}$ receives given a fixed $b \in \{0, 1\}$. When it is first invoked (step 1), it produces $PK = (N, e)$ and some string aux. Next (step 2), $\mathtt{msg}_0$ and $\mathtt{msg}_1$ are selected by the challenger by running $\mathcal{M}(1^k, PK, aux)$; let $x_0 = \mathsf{Hash}(\mathtt{msg}_0)$ and $x_1 = \mathsf{Hash}(\mathtt{msg}_1)$. Let r_0 and r_1 be sampled at random from $\mathbb{Z}_N^*$, and let $z_0 = x_0 r_0^e$ and $z_1 = x_1 r_1^e$ be the blinded messages the challenger sends to $\mathcal{A}$ (step 3), and let s_0 and s_1 be the values $\mathcal{A}$ sends in return — the order in which they are sent depends on b (step 4). Next (step 5) if $s_0^e = z_0$ and $s_1^e = z_1$, the challenger computes $\sigma_0 = s_0/r_0$ and σ_1/r_1 and sends to the adversary the values $(\mathtt{msg}_0, \sigma_0)$ and $(\mathtt{msg}_1, \sigma_1)$, else it sends it $\perp$.

Consider an alternative pair of experiments for $b \in \{0, 1\}$; here the challenger is computationally unbounded. The challenger begins by selecting $\mathtt{msg}_0$ and $\mathtt{msg}_1$ from $\mathcal{M}(1^k, PK, aux)$. We have two cases: Case A, in which there exist (σ_0, σ_1) such that $\sigma_0^e = \mathsf{Hash}(\mathtt{msg}_0)$ and $(\sigma_1)^e = \mathsf{Hash}(\mathtt{msg}_1)$; and Case B, in which the pair exist (σ_0, σ_1) does not exist. Since this challenger is unbounded, it identifies which case it is in, and acts as follows:

In Case A, in Step 2, the challenger picks z and z^* uniformly at random from $\mathbb{Z}_N^*$ and sends (z, z^*) to $\mathcal{A}$. It receives (s, s^*). In step 5, if $s^e \neq z$ or $(s^*)^e \neq (z^*)^e$, then it sends $\perp$ to $\mathcal{A}$. Else, it samples valid signatures σ_0 and σ_1 for $\mathtt{msg}_0$ and $\mathtt{msg}_1$, respectively, and sends to the adversary $\mathcal{A}$ the values $(\mathtt{msg}_0, \sigma_0)$ and $(\mathtt{msg}_1, \sigma_1)$.

In Case B, the challenger follows the protocol.

It is easy to see that, in the alternative experiment, the adversary's view is independent of b. To see that the alternative experiment gives $\mathcal{A}$ a view that's identical to the blind token game in Case A, note that the challenger choosing $r_0 = s/\sigma_0$ and $r_1 = s^*/\sigma_1$ in step 3 corresponds to having $b = 0$ in the blind token game, while choosing $r_0 = s^*/\sigma_0$ and $r_1 = s/\sigma_1$ corresponds to $b = 1$. Since r_0 and r_1 are chosen uniformly at random, the two options are equally likely. In Case B, since one or both signatures don't exist, the adversary's view is independent of b as well, since the pair of messages $(\mathtt{msg}_0, \mathtt{msg}_1)$ is just as likely as $(\mathtt{msg}_1, \mathtt{msg}_2)$. □

5 Unforgeability of RSA-BSSA

Recall that an algorithm $\mathcal{A}$ is said to break the security of a cryptographic scheme *in the random-oracle model* [7] if its success probability is non-negligible when a specific component of the scheme, typically a hash function, is replaced by a random oracle. Security in the random-oracle model means that no polynomial-time algorithm can break the scheme in the random-oracle model.

A proof of security in the random-oracle model does not, in fact, imply a proof of security in the plain model (i.e. where no component of the scheme is modeled as a random oracle) [12]. However, it is considered evidence of security that's good enough in practice.

In a random-oracle-based reduction, the reduction is typically privy to all the hash function queries the adversary issues. Another privilege that such a reduction has (in the standard, so-called "programmable" random-oracle model — these different flavors are explored by Fischlin et al. [19]) is that it can answer such a query with any value it desires. Since the adversary expects the answers to its queries to be truly random, as long as the reduction's responses are distributed at random (or are indistinguishable from random), the adversary's success probability will be as high when interacting with the reduction as when attacking the scheme.

We will prove strong one-more unforgeability of RSA-BSSA in the random-oracle model under the one-more-RSA assumption introduced by Bellare, Namprempre, Pointcheval and Semanko [6]. They also showed that the one-more-RSA assumption (stated formally in Appendix A) holds if an only if the following problem, called the alternative chosen-target RSA inversion (RSA-ACTI) problem, is hard:

Definition 8 (RSA-ACTI [6]). *Let $\mathcal{A}$ be an oracle Turing machine. For the security parameter k, let the experiment $\mathbf{Exp}^{rsa-acti}_{\mathcal{A}}(k)$ be defined as follows:*

RSA key pair generation *The challenger generates an RSA public key (N, e) and secret key d corresponding to the security parameter k. Let us define the following oracles:*
1. *The RSA inversion oracle $\mathcal{O}_I(\cdot, N, d)$ that, on input $y \in \mathbb{Z}_N^*$, returns x such that $x^e = y \bmod N$; i.e., it returns $y^d \bmod N$ where $ed \equiv 1 \bmod \varphi(N)$.*
2. *An oracle $\mathcal{O}_R(\cdot, N)$ that, when queried, issues a random RSA inversion challenge point, i.e. a random element of $\mathbb{Z}_N^*$. By y_i, let us denote the outcome of the i^{th} such query.*

$\mathcal{A}$ is invoked *The challenger invokes $\mathcal{A}^{\mathcal{O}_I(\cdot,N,d),\mathcal{O}_R(\cdot,N)}(N, e)$ and responds to its oracle queries. Eventually, $\mathcal{A}$ terminates.*

$\mathcal{A}$'s success criterion *Let ℓ be the number of queries $\mathcal{A}$ issued to $\mathcal{O}_I(\cdot, N, d)$. Let $(y_1, \ldots, y_n)$ be the values $\mathcal{A}$ received from $\mathcal{O}_R(\cdot, N)$. Let $(z_1, \ldots, z_n)$ be $\mathcal{A}$'s output. For $1 \leq i \leq n$, z_i is correct if $z_i^e = y_i \bmod N$. $\mathcal{A}$ is successful if $|\{i \ : \ z_i \text{ is correct}\}| \geq \ell + 1$.*

By $\mathbf{Adv}^{rsa-acti}_{\mathcal{A}}(k)$ we denote the probability that $\mathcal{A}$ is successful in $\mathbf{Exp}^{rsa-acti}_{\mathcal{A}}(k)$. The RSA-ACTI problem is hard if for any probabilistic polynomial-time $\mathcal{A}$, $\mathbf{Adv}^{rsa-acti}_{\mathcal{A}}(k)$ is negligible.

Theorem 6. *Let $\mathcal{A}$ be an algorithm that breaks strong one-more unforgeability of the basic RSA-BSSA scheme (Definition 4) where both $\mathsf{Hash}(\cdot, \ell)$ and $\mathsf{MGF}(\cdot, \ell)$ are random oracles for every integer ℓ. Let $t_{\mathcal{A}}(k)$ be an upper bound on its running time; let $p_{\mathcal{A}}(k)$ be its success probability.*

Then there exists an algorithm $\mathcal{B}$ that solves the RSA-ACTI problem (Definition 8) in $t^{\mathcal{B}}(k) = O(poly(k) + t^{\mathcal{A}}(k))$ time with probability $p_{\mathcal{B}}(k) = p_{\mathcal{A}}(k) - \Theta(t^2_{\mathcal{A}}(k)2^{-8hLen})$.

This theorem, i.e. the unforgeability of the *basic* RSA-BSSA scheme, implies unforgeability of variants A and B. For variant A, the additional proof that's part of the public key can be simulated in the random-oracle model as shown by Goldberg et al. [22]. For variant B, unforgeability follows from that of the basic scheme, since a signature in variant B on message `msg` with randomness `rand` is also a signature in the basic scheme on message `msg` $\circ$ `rand`.

Corollary 3. *Let $\mathcal{A}$ be an algorithm that breaks strong one-more unforgeability of RSA-BSSA variants A or B (Definition 4) where both $\mathsf{Hash}(\cdot, \ell)$ and $\mathsf{MGF}(\cdot, \ell)$ are random oracles for every integer ℓ. Let $t_{\mathcal{A}}(k)$ be an upper bound on its running time; let $p_{\mathcal{A}}(k)$ be its success probability.*

Then there exists an algorithm $\mathcal{B}$ that solves the RSA-ACTI problem (Definition 8) in $t^{\mathcal{B}}(k) = O(poly(k) + t^{\mathcal{A}}(k))$ time with probability $p_{\mathcal{B}}(k) = p_{\mathcal{A}}(k) - \Theta(t^2_{\mathcal{A}}(k)2^{-8hLen})$.

Proof. (of Theorem 6) Let $q^{\mathcal{A}}_{\mathsf{Hash}}(PK; R)$, $q^{\mathcal{A}}_{\mathsf{MGF}}(PK; R)$ and $q^{\mathcal{A}}_{\mathsf{BSig}}(PK; R)$ be the number of queries $\mathcal{A}$ makes to Hash, MGF and BSig respectively when interacting with it challenger on input a specific public key PK; R denotes the randomness of the experiment (i.e., both the random tape of $\mathcal{A}$ and that of the challenger). When PK and R are clear from context, we will write $q^{\mathcal{A}}_{\mathsf{Hash}}$, $q^{\mathcal{A}}_{\mathsf{MGF}}$ and $q^{\mathcal{A}}_{\mathsf{BSig}}$.

Without loss of generality, let us assume that $\mathcal{A}$'s output is either empty (i.e., $\mathcal{A}$ fails to win the game) or consists solely of $q^{\mathcal{A}}_{\mathsf{BSig}} + 1$ message-signature pairs that pass verification. Let us call such an $\mathcal{A}$ a "high-achieving adversary" in the sequel. The reason we can assume that $\mathcal{A}$ is high-achieving is that, if it's not, we could modify $\mathcal{A}$ into an algorithm $\mathcal{A}'$ that verifies $\mathcal{A}$'s output and, if $\mathcal{A}$ succeeded, outputs the first $q^{\mathcal{A}}_{\mathsf{BSig}} + 1$ pairs that pass verification. By definition of one-more unforgeability, $\mathcal{A}'$ succeeds with the same probability as $\mathcal{A}$, and has a comparable running time.

We will construct the reduction $\mathcal{B}$ that will use a high-achieving $\mathcal{A}$ as a subroutine.

Input to the reduction. The reduction plays the role of the attacker in experiment $\mathbf{Exp}^{rsa-acti}_{\mathcal{B}}(k)$. Thus, it takes as input the RSA public key (N, e) that had been generated by RSA's key generation on input the security parameter k.

The oracle the reduction may use. As described in Definition 8, $\mathcal{B}$ has access to two oracles:

1. The RSA inversion oracle $\mathcal{O}_I(\cdot, N, d)$ that, on input $y \in \mathbb{Z}^*_N$, returns x such that $x^e = y \bmod N$; i.e., it returns $y^d \bmod N$ where $ed \equiv 1 \bmod \varphi(N)$.
2. An oracle $\mathcal{O}_R(\cdot, N)$ that, when queried, issues a random RSA inversion challenge point, i.e. a random element of $\mathbb{Z}^*_N$. By y_i, let us denote the outcome of the i^{th} such query.

How the reduction interacts with $\mathcal{A}$. The adversary $\mathcal{A}$ is attacking the strong one-more unforgeability property of the RSA-BSSA scheme as described in Definition 4. Thus, $\mathcal{A}$ will need to receive PK as input; the reduction sets $PK = (N, e)$, where (N, e) is its own input. Since the reduction is in the random-oracle model, $\mathcal{A}$ will expect oracle access to Hash and MGF, which $\mathcal{B}$ will respond to as described below. $\mathcal{A}$ will also engage with the signer in the blind signing protocol; in the case of RSA-BSSA, this will involve oracle access to $\mathsf{BSig}(SK, \cdot)$; below, we describe how $\mathcal{B}$ will handle this as well. Finally, $\mathcal{A}$ terminates and produces some output; below, we describe how $\mathcal{B}$ uses $\mathcal{A}$'s output to compute a solution to the RSA-ACTI problem.

How the reduction will handle $\mathcal{A}$'s queries to $\mathsf{Hash}(\cdot, \cdot)$. A relevant query to $\mathsf{Hash}(\cdot, \cdot)$ is (v, ℓ) such that $\ell = hLen$ and the first eight bytes of v are all 0.

Let (v, ℓ) be a query that is *not* relevant. A value v that is derived as part of signature verification must begin with 64 0s; if v is not of that form, we know that we will never encounter the need to calculate $\mathsf{Hash}(v, hLen)$ as part of verifying a signature. (A detailed description of the signature verification algorithm provided in Appendix C clarifies this point; see Step 6.) We also know that for any length $\ell \neq hLen$, $\mathsf{Hash}(v, \ell)$ is not computed as part of signature verification. Thus, there is no need to prepare a response to this query in any special way. In response to (v, ℓ), the reduction returns a randomly sampled string h of ℓ bytes and stores $((v, \ell), h)$ for future reference.

In contrast, the response to the i^{th} relevant query is set up in such a way that, should that query be part of the successful verification of one of the signatures returned by the adversary, it should allow the reduction to invert RSA at a challenge point y_i.

More precisely: Let the i^{th} relevant query to Hash be the pair $(v_i, hLen)$. Parse v_i as follows: $v_i = 0^{64} \circ \mathtt{mHash}_i \circ \mathtt{salt}_i$. Implicitly, $\mathtt{mHash}_i$ is computed from some unknown M_i, and $\mathtt{salt}_i$ are the last *lenSalt* bits of v_i.

Our goal is to ensure that, if the adversary ever returns (M, σ) that passes the verification algorithm such that $\mathtt{mHash} = \mathtt{mHash}_i$, and $\mathtt{salt} = \mathtt{salt}_i$, then $\sigma^e = y_i r_i^e \bmod N$ for some challenge point y_i and a value r_i known to the reduction. Then the reduction can invert RSA at y_i by outputting σ / r_i.

First, the reduction obtains the challenge y_i by querying $\mathcal{O}_R(\cdot, N)$. Next, it samples from $\mathbb{Z}_N^*$ until it finds a value r_i such that, for $w_i = y_i r_i^e$, $w_i < 2^{k-1}$ (i.e. $k - 1$ bits are sufficient to encode it) and the binary representation of w_i ends in the byte $0xBC$. In expectation, it will take between 256 and 512 attempts to find such r_i, depending on how close N is to 2^k: at least half the time, $w_i < 2^{k-1}$, and conditioned on that, it starts with $0xBC$ one in every 256 tries.

Next, execute the following steps that determine how to fix MGF in one point, and what value to return in response to this query. The goal is to ensure that, if (M, σ) is a message-signature pair accepted by the verification algorithm, and $v_i = 0^{64} \circ \mathtt{mHash} \circ \mathtt{salt}$ is the input to Hash computed as part of verification (i.e. it is the value M' computed in Step 6 and queried in Step 7 of the detailed verification algorithm in Appendix C), then $\sigma^e \bmod N = w_i$.

1. Set $\mathtt{EM}_i = \mathsf{I2OSP}(w_i, emLen)$ (recall that $emLen = \lceil (k-1)/8 \rceil$).

2. Parse $\texttt{EM}_i = \texttt{maskedDB}_i \circ H_i \circ 0xBC$ (as described in Step 2 of the verification procedure in Appendix C). Since the reduction used the sampling procedure above to obtain w_i, $\texttt{EM}_i$ ends in the byte $0xBC$.
3. In order to ensure that $\texttt{DB}_i$ that will be computed in Step 4 (of the detailed verification procedure) contains the same salt value as v_i, carry out the following steps:
   – Let $\texttt{mHash}_i$ and $\texttt{salt}_i$ be the strings of $hLen$ and $sLen$ bytes, respectively, such that $v_i = 0^{64} \circ \texttt{mHash}_i \circ \texttt{salt}_i$.
   – Let $\texttt{DB}_i = 0^a \circ 0x01 \circ \texttt{salt}_i$, where $a = 8(lenDB - 1 - sLen)$.
   – Let $\texttt{dbMask}'_i = \texttt{DB}_i \oplus \texttt{maskedDB}_i$. Note that $\texttt{dbMask}'_i$ must start with 0^p, since $\texttt{DB}_i$ starts with 0s, and the fact that $\texttt{maskedDB}$ is the beginning of the string output by $\mathsf{I2OSP}$ ensures that it begin with p 0s. Let $\texttt{dbMask}_i$ be the result of replacing the first p bits of $\texttt{dbMask}'_i$ with random bits.
   – If $\mathsf{MGF}(H_i, lenDB)$ is already defined, fail. Else, set it to the value $\texttt{dbMask}_i$.
4. Set $\mathsf{Hash}(v_i, hLen) = H_i$, save (i, v_i, w_i, r_i, H_i) for future reference.

Return H_i.

How the reduction will handle $\mathcal{A}$'s queries to $\mathsf{MGF}(\cdot,\cdot)$. Let u be a query to MGF. Case 1: $\mathsf{MGF}(u, \ell)$ is already fixed as a result of a previous query to Hash or MGF; then return the value $\mathsf{MGF}(u, \ell)$. Case 2: $\mathsf{MGF}(u, \ell)$ is not yet fixed; then return a random string of ℓ octets.

How the reduction will handle $\mathcal{A}$'s queries to $\mathsf{BSig}(SK, \cdot)$. Once `blinded_msg` from the adversary $\mathcal{A}$ is received, compute $m = \mathsf{OS2IP}(\texttt{blinded\_msg})$. If $m \geq N$, then it fails. Otherwise, it sends m to its RSA inversion oracle $\mathcal{O}_I(\cdot, N, d)$. In turn, $\mathcal{O}_I(m, N, d)$ returns s such that $s^e = m \bmod N$. $\mathcal{B}$ computes $\texttt{blinded\_sig} = \mathsf{I2OSP}(s, kLen)$ and returns it to $\mathcal{A}$.

How the reduction will process $\mathcal{A}$'s output. At the end of its execution, the adversary $\mathcal{A}$ outputs a set of message-signature pairs $\{(M_j, \sigma_j)\}$. First, $\mathcal{B}$ verifies these message-signature pairs, as follows: it runs the verification algorithm as described in Appendix C. When the verification algorithm queries Hash and MGF for a value previously queried by $\mathcal{A}$, Hash and MGF return the same string as was returned to $\mathcal{A}$. When the verification algorithm queries Hash and MGF for a value not previously queried by $\mathcal{A}$, Hash and MGF return random strings.

Next, $\mathcal{B}$ fails if some message-signature pair (M_j, σ_j) is accepted by the verification algorithm, and yet $\mathcal{A}$ had not made the queries to Hash and MGF that the verification algorithm just made when verifying (σ_j, M_j).

Else, $\mathcal{B}$ proceeds as follows. Recall that n is the number of queries that $\mathcal{B}$ has made to its challenge oracle, i.e. the number of challenge points y_j that $\mathcal{B}$ has received. For $1 \leq i \leq n$, initialize $z_i = \perp$. Next, for each j such that (M_j, σ_j) is accepted by the verification algorithm, find i such that $\sigma_j^e \bmod N \equiv w_i$. (If no such i exists, fail.) Then $\sigma_j^e = y_i r_i^e \bmod N$, so $y_i = (\sigma_j / r_i)^e$ and so $\mathcal{B}$ has inverted RSA at y_i; set $z_i = \sigma_j / r_i$.

$\mathcal{B}$ outputs $z_1, \ldots, z_n$.

Analysis of the reduction. To conclude our proof of security, we need to prove the following three claims. First, we show that an adversary that wins the strong one-more-forgery game against RSA-BSSA must (other than with very small probability) query MGF and Hash for all the values that will be queried over the course of the verification of its signatures. This will allow us to assume that we are dealing with the adversary that always makes these queries prior to outputting its signatures and makes sure that verification accepts; we will call such an adversary a "make-sure" adversary. More formally:

Claim 7 *Let $\mathcal{A}$ be a high-achieving adversary that wins the strong one-more-forgery game against RSA-BSSA in the random-oracle model with probability $p_{\mathcal{A}}(k)$. Let E be the event that $\mathcal{A}$ wins the game and for each of the message-signature pairs it produced, it issued the queries to both* $\mathsf{MGF}(\cdot, lenDB)$ *and* $\mathsf{Hash}(\cdot, hLen)$ *needed for verifying the message-signature pair some time during the course of its execution. Then* $\Pr[E] \geq p_{\mathcal{A}}(k) - \Theta(t^2_{\mathcal{A}}(k) 2^{-8hLen})$.

Next, we show that for a "make-sure" $\mathcal{A}$, our reduction succeeds. This is done in two steps: (1) showing that the view that the reduction provides for $\mathcal{A}$ is identical to its view in the one-more-forgery game in the random-oracle model; (2) showing that whenever a "make-sure" $\mathcal{A}$ is successful, the reduction succeeds in the RSA-ACTI game. More formally:

Claim 8 *Let $\mathcal{A}$ be any adversary. In the random-oracle model, the view that $\mathcal{A}$ receives when interacting with the strong one-more unforgeability game challenger for RSA-BSSA is identical to the one $\mathcal{A}$ obtains in an interaction with the reduction $\mathcal{B}$ whenever $\mathcal{B}$ does not fail while answering $\mathcal{A}$'s queries. Moreover, the probability that $\mathcal{B}$ fails while answering a query from $\mathcal{A}$ is $O(t^2(k) 2^{-8hLen})$.*

Claim 9 *Let $\mathcal{A}$ be an adversary that wins the strong one-more unforgeability game against RSA-BSSA in the random-oracle model with probability $p_{\mathcal{A}}(k)$. Then $\mathcal{B}$ wins the RSA-ACTI game with probability $p_{\mathcal{A}}(k) - \Theta(t^2_{\mathcal{A}}(k) 2^{-8hLen})$.*

The proofs of Claims 7, 8 and 9 are in Appendix D. □

Acknowledgments

I thank Frederic Jacobs for numerous discussions about RSA-BSSA, and Chris Wood and Steve Myers for helpful feedback. I am also grateful to the anonymous referees for constructive comments. This paper was supported by Apple Inc. I also acknowledge the support of NSF awards #2154170 and #2154941.

References

1. Michel Abdalla, Chanathip Namprempre, and Gregory Neven. On the (im)possibility of blind message authentication codes. In David Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *LNCS*, pages 262–279. Springer, Heidelberg, February 2006.
2. Masayuki Abe. A secure three-move blind signature scheme for polynomially many signatures. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 136–151. Springer, Heidelberg, May 2001.

3. Masayuki Abe and Tatsuaki Okamoto. Provably secure partially blind signatures. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 271–286. Springer, Heidelberg, August 2000.
4. Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 1087–1098. ACM Press, November 2013.
5. Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The power of RSA inversion oracles and the security of Chaum's RSA-based blind signature scheme. In Paul F. Syverson, editor, *FC 2001*, volume 2339 of *LNCS*, pages 319–338. Springer, Heidelberg, February 2002.
6. Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, June 2003.
7. 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*, pages 62–73. ACM Press, November 1993.
8. Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 399–416. Springer, Heidelberg, May 1996.
9. Mihir Bellare and Phillip Rogaway. PSS: Provably secure encoding method for digital signatures. Submission to IEEE P1363, 1998.
10. Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 33–53. Springer, Heidelberg, October 2021.
11. Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Heidelberg, January 2003.
12. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998.
13. David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982.
14. David Chaum. Blind signature systems. In *CRYPTO '83*, pages 153–156. Plenum, 1983.
15. David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 319–327. Springer, Heidelberg, August 1990.
16. Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 272–287. Springer, Heidelberg, April / May 2002.
17. IETF Draft. Denis, F. and F. Jacobs and C.A. Wood, "RSA blind signatures", February 2022. `https://datatracker.ietf.org/doc/draft-irtf-cfrg-rsa-blind-signatures/`.
18. IETF Draft. Denis, F. and F. Jacobs and C.A. Wood, "RSA blind signatures", March 2021. `https://datatracker.ietf.org/doc/html/draft-wood-cfrg-rsa-blind-signatures-00`.

19. Marc Fischlin, Anja Lehmann, Thomas Ristenpart, Thomas Shrimpton, Martijn Stam, and Stefano Tessaro. Random oracles with(out) programmability. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 303–320. Springer, Heidelberg, December 2010.
20. Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind schnorr signatures and signed ElGamal encryption in the algebraic group model. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 63–95. Springer, Heidelberg, May 2020.
21. Steven D. Galbraith, John Malone-Lee, and Nigel P. Smart. Public key signatures in the multi-user setting. *Inf. Process. Lett.*, 83(5):263–266, 2002.
22. Sharon Goldberg, Leonid Reyzin, Omar Sagga, and Foteini Baldimtsi. Efficient noninteractive certification of RSA moduli and beyond. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 700–727. Springer, Heidelberg, December 2019.
23. Shafi Goldwasser, Silvio Micali, and Ronald Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.
24. Eduard Hauck, Eike Kiltz, and Julian Loss. A modular treatment of blind signatures from identification schemes. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 345–375. Springer, Heidelberg, May 2019.
25. Eduard Hauck, Eike Kiltz, Julian Loss, and Ngoc Khanh Nguyen. Lattice-based blind signatures, revisited. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 500–529. Springer, Heidelberg, August 2020.
26. Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of blind digital signatures (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 150–164. Springer, Heidelberg, August 1997.
27. Jonathan Katz and Nan Wang. Efficiency improvements for signature schemes with tight security reductions. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM CCS 2003*, pages 155–164. ACM Press, October 2003.
28. Anna Lysyanskaya. Security analysis of RSA-BSSA. *IACR Cryptol. ePrint Arch.*, page 895, 2022.
29. David Pointcheval and Jacques Stern. Provably secure blind signature schemes. In Kwangjo Kim and Tsutomu Matsumoto, editors, *ASIACRYPT'96*, volume 1163 of *LNCS*, pages 252–265. Springer, Heidelberg, November 1996.
30. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000.
31. IETF RFC3447. Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", February 2003. `https://datatracker.ietf.org/doc/html/rfc3447`.
32. IETF RFC8017. Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", November 2016. `https://datatracker.ietf.org/doc/html/rfc8017`.
33. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991.
34. Claus-Peter Schnorr. Security of blind discrete log signatures against interactive attacks. In Sihan Qing, Tatsuaki Okamoto, and Jianying Zhou, editors, *ICICS 01*, volume 2229 of *LNCS*, pages 1–12. Springer, Heidelberg, November 2001.

35. Dominique Schröder and Dominique Unruh. Security of blind signatures revisited. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 662–679. Springer, Heidelberg, May 2012.
36. Victor Shoup. *A Computational Introduction to Number Theory and Algebra (2nd Edition)*. Cambridge University Press, 2009.
37. Stefano Tessaro and Chenzhi Zhu. Short pairing-free blind signatures with exponential security. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 782–811. Springer, Heidelberg, May / June 2022.

A Statement of computational hardness assumptions

Bellare, Namprempre, Pointcheval and Semanko [5, 6] introduced the RSA known-target inversion problem (RSA-KTI) defined below; the definition we give here is identical to theirs:

Definition 9 (Known-Target Inversion Problem: RSA-KTI [6]). *Let $\mathcal{A}$ be an oracle Turing machine. For the security parameter k and any function $m\colon \mathbb{N} \mapsto \mathbb{N}$, let the experiment $\mathbf{Exp}^{rsa-kti}_{\mathcal{A},m}(k)$ be defined as follows:*

RSA key pair generation *The challenger generates an RSA public key (N, e) and secret key d corresponding to the security parameter k. Let $\mathcal{O}_I(\cdot, N, d)$ be the RSA inversion oracle; i.e., on input $y \in \mathbb{Z}^*_N$, it returns $x = y^d \bmod N$.*

Challenge values are selected *For $1 \leq i \leq m(k) + 1$, pick $y_i \leftarrow \mathbb{Z}^*_N$.*

$\mathcal{A}$ is invoked *The challenger invokes $\mathcal{A}^{\mathcal{O}_I(\cdot,N,d)}(N, e, k, y_1, \ldots, y_{m(k)+1})$ and responds to its oracle queries. Eventually, $\mathcal{A}$ terminates.*

$\mathcal{A}$'s success criterion *$\mathcal{A}$ is successful if (1) it issued no more than $m(k)$ queries to $\mathcal{O}_I(\cdot, N, d)$; and (2) $\mathcal{A}$ output is $(z_1, \ldots, z_{m(k)+1})$ such that, for all $1 \leq i \leq m(k) + 1$, $z_i^e = y_i \bmod N$.*

By $\mathbf{Adv}^{rsa-kti}_{\mathcal{A},m}(k)$ we denote the probability that $\mathcal{A}$ is successful in $\mathbf{Exp}^{rsa-kti}_{\mathcal{A},m}(k)$. The RSA-KTI[m] problem is hard if for any probabilistic polynomial-time $\mathcal{A}$, $\mathbf{Adv}^{rsa-kti}_{\mathcal{A},m}(k)$ is negligible; the RSA-KTI problem is hard if the RSA-KTI[m] problem is hard for any polynomially bounded m.

Assumption A1 (One-more-RSA Assumption [6]) *The known-target inversion problem RSA-KTI is intractable.*

Bellare et al. then reduced breaking the assumption (i.e., solving RSA-KTI) to solving the seemingly easier RSA-ACTI problem stated in Definition 8. (See Theorems 4.1 and 5.4 in Bellare et al. [6].) Thus, to prove security of the scheme, it is sufficient to give a polynomial-time reduction that breaks RSA-ACTI with access to an adversary $\mathcal{A}$ attacking the scheme.

B Background on RSA-PSS

Bellare and Rogaway [8, 9] introduced the probabilistic signature scheme (PSS) which is a variant of the RSA signature that has a tight security reduction in the random-oracle model; the reduction also remains tight for the Rabin trapdoor permutation. This popular signature scheme has been standardized, and is used extensively. Follow-up work [27] has shown a tight reduction for additional variants of PSS; however, the follow-ups have not been standardized.

There are a number of disconnects between the RSA-PSS paper [8] and RSA-PSS as described in the PKCS#1 standards [31, 32]. Some of them are covered in the standards documents themselves, but some do not appear to have received attention. Let us highlight them here for completeness:

The message M vs. its hashed representation `mHash`. Bellare and Rogaway introduced two versions of PSS. In the first version, which we will call BR-PSS (they just call it PSS, but we add the prefix "BR" to avoid confusion with the IETF's PKCS#1 standard with the same name), M can be any binary string, and a signature verification algorithm takes as input both M and the signature σ. The innovation of this version of PSS is a security proof that is tighter than previously known proofs for previously known variations of RSA signatures; it was also a convenient building block for analyzing their second proposal, BR-PSS-R (again, they just call it PSS-R, but we add the prefix "BR" to clarify that it came from Bellare and Rogaway and not from the IETF PSS standard).

BR-PSS-R is the more interesting scheme, as it allows for message recovery. Here, the message M needs to be a binary string of a fixed length that was not to exceed $k - k_0 - k_1 - 1$, where k is the length of the modulus, $k_0 = 8\mathit{lenSalt}$ the length of the salt, and $k_1 = 8\mathit{hLen}$ the length of the output of a hash function. In BR-PSS-R, the signature σ alone contains all the information needed to both recover M and verify that it was signed.

Bellare and Rogaway's analysis showed both BR-PSS and BR-PSS-R to be secure in the random-oracle model under the RSA assumption: an attacker who succeeded in attacking BR-PSS or BR-PSS-R in time $t(k)$ with probability $\epsilon(k)$ could be plugged into a reduction that solves RSA in time $O(\mathit{poly}(k)+t(k))$ with probability $\epsilon(k) - \mathit{poly}(k)t^2(k)(2^{-k_1} + 2^{-k_0})$. (Note that Bellare and Rogaway's analysis is somewhat more fine-grained, since it considers not just $\mathcal{A}$'s running time but also the number of oracle queries it makes; the discussion above is a simplified statement of their findings.)

The PKCS#1 standard version of PSS, which we will call PKCS-PSS, deviates from the Bellare-Rogaway proposals. It allows for the message to be any string, so it would seem that the BR-PSS would be a good scheme to incorporate. But, unlike BR-PSS whose first step is to compute $\mathsf{Hash}(M, \mathit{salt})$ for a randomly chosen string *salt* of k_0 bits, the first step of PKCS-PSS is to hash the message M down to $\texttt{mHash} = \mathsf{Hash}(M)$ of *hLen* bytes.

It may, of course, sound concerning that PKCS-PSS is sufficiently different from BR-PSS that the latter's analysis no longer applies. However, PKCS-PSS is (essentially) the same as BR-PSS-R where `mHash` is the message that's signed. Thus, Bellare and Rogaway's analysis for BR-PSS-R applies (essentially) to PKCS-PSS.

The length of the salt. PKCS-PSS is well-defined even when the length of the salt varies, from 0 bytes to *hLen* bytes. The original analysis of BR-PSS does not apply when the salt is too short: the reduction aborts in the event that a value it picked for the salt value ever repeats, which will happen non-negligibly often if the salt is too short. Follow-up work [16, 27] showed that the resulting scheme was still secure with a (moderately) tight reduction.

Other differences. As pointed out in the IETF documents [31, 32], there are fixed substrings added to the PKCS-PSS encodings at several stages.

C The verification algorithm, step by step

For the security analysis, it is helpful to recall all the steps that the signature verification algorithm will take (rather than deferring to subroutines that are defined elsewhere). The notation $0xUV$, where U and V are hexadecimal digits, denote the value of an octet, or byte; e.g., $0x3a$ corresponds to the binary string 00111100. The symbol $\circ$ denotes concatenation. Using the PSS encoding from the PKCS#1 standard [31, 32], verifying a signature σ for a message M consists of the following steps (note: these steps are equivalent to those in the PKCS# standard, but not described in exactly the same way):

1. Compute the encoded message $\texttt{EM} = \mathsf{I2OSP}(\sigma^e \bmod N, \mathit{emLen})$. Specifically, $\mathsf{I2OSP}$ will reject if $\sigma^e \bmod N$ is greater than $2^{8\mathit{emLen}}$; else, it outputs $\mathit{emLen} = \lceil (k-1)/8 \rceil$ octets that, when viewed as a binary integer, equal $\sigma^e \bmod N$. Note that, whenever $k-1$ is not a multiple of 8, this will always result in having $\texttt{EM}$ (viewed as a bit string) start with up to 7 zeroes. Let $0 \le p \le 7$ be such that for maximal positive integer m, $k-1 = 8m + (8-p)$. I.e. p is the number of extra bits we get when converting the bit representation of a $k-1$-but integer into the byte representation of the same integer.
2. If $\texttt{EM}$ doesn't end in the byte $0xBC$, reject. Else, parse $\texttt{EM}$ as follows: the first $\mathit{lenDB} = \mathit{emLen} - \mathit{hLen} - 1$ bytes are the string $\texttt{maskedDB}$; the next hLen bytes are the string H, and the last byte, as we already know, is $0xBC$. To summarize, $\texttt{EM} = \texttt{maskedDB} \circ H \circ 0xBC$.
3. Let $\texttt{dbMask} = \mathsf{MGF}(H, \mathit{lenDB})$.
4. Let $\texttt{DB}' = \texttt{maskedDB} \oplus \texttt{dbMask}$; let $\texttt{DB}$ be the same string as $\texttt{DB}'$ except that the first p bits are set to 0. (This is because, since we set p to be $0 \le p \le 7$ be such that for some integer m, $k-1 = 8m + (8-p)$, the first p bits of the byte encoding of a $k-1$-bit integer are always 0, so the value we "unmask" starts at bit $p+1$.)
5. If $\texttt{DB}$ does not start with $\mathit{lenDB} - 1 - \mathit{sLen}$ $0x00$ octets followed by $0x01$, then reject. Else, let $\texttt{salt}$ be the last sLen octets of $\texttt{DB}$. To summarize, $\texttt{DB} = 0x00 \ldots 0x00 \circ 0x01 \circ \texttt{salt}$.

6. Let $M' = 0^{64} \circ \texttt{mHash} \circ \texttt{salt}$, where $\texttt{mHash} = \mathsf{CRHF}(M)$. (As usual, by 0^{64} we denote a binary string of 64 zeroes; we can also think of it as a string of eight bytes, each set to $0x00$.)
7. If $H = \mathsf{Hash}(M', hLen)$, accept, else, reject.

D Proofs of Claims 7, 8 and 9

D.1 Proof of Claim 7

Recall that we are dealing a high-achieving $\mathcal{A}$ whose output is always either empty or consists of exactly $q^{\mathcal{A}}_{\mathsf{BSig}} + 1$ message-signature pairs that pass verification, where $q^{\mathcal{A}}_{\mathsf{BSig}}$ is the number of signing queries made by $\mathcal{A}$.

Let F be the event that $\mathcal{A}$ succeeds and among these message-signature pairs, there is (σ, M) such that $\mathcal{A}$ had not made the queries to one or both of $\mathsf{MGF}(\cdot, lenDB)$ and $\mathsf{Hash}(\cdot, hLen)$ that the signature verification algorithm makes when verifying (σ, M). Since $\Pr[E] = p_{\mathcal{A}}(k) - \Pr[F]$, it is sufficient to derive an upper bound on $\Pr[F]$.

Let $\texttt{EM} = \texttt{maskedDB} \circ H \circ 0xBC$, $\texttt{dbMask}$, $\texttt{DB}$, $\texttt{salt}$ and M' be the values derived when verifying (σ, M). Consider the following cases.

Case 1: $\mathcal{A}$ has queried MGF on input $(H, lenDB)$ but $\mathsf{Hash}(M', hLen)$ was never queried by $\mathcal{A}$. Then the probability that $\mathsf{Hash}(M', hLen) = H$, leading the verification algorithm to accept, is $p_1 = 2^{-8hLen}$.

Case 2: $\mathcal{A}$ has not queried MGF on input $(H, lenDB)$, and H is a value that, over the course of $\mathcal{A}$'s execution, has never been the output of $\mathsf{Hash}(\cdot, hLen)$ on input a relevant query (recall that a query is relevant if it is a pair (v, ℓ) such that $\ell = hLen$ and the first eight bytes of v are all 0). Then the probability that $H = \mathsf{Hash}(M', hLen)$ is at most $p_2 = 2^{-8hLen}$, because either $\mathcal{A}$ has already queried $\mathsf{Hash}(M', hLen)$ in which case we know the output was not H, or it has not, in which case it is H with probability exactly 2^{-8hLen}.

Case 3: $\mathcal{A}$ has not queried MGF on input $(H, lenDB)$, and $H = \mathsf{Hash}(a, hLen)$ for some previous relevant query $(a, hLen)$ to Hash. Let us estimate the probability that the verification algorithm accepts. There are two disjoint possibilities:

Case 3a: a is unique, i.e., no other query to $\mathsf{Hash}(\cdot, hLen)$ gave rise to the output H. Let us parse a as follows: $a = 0^{64} \circ \texttt{mHash}_a \circ \texttt{salt}_a$ (we can parse it this way because it is a relevant query). Let $\texttt{DB}_a = 0^{8(lenDB-1-sLen)} \circ 0x01 \circ \texttt{salt}_a$. Let $\texttt{dbMask}'_a = \texttt{DB}_a \oplus \texttt{maskedDB}$, where recall that $\texttt{maskedDB}$ is the value that the verification algorithm obtains as a result of parsing $\texttt{EM}$ in Step 2 of the detailed verification procedure (see Appendix C). The verification algorithm will accept in this case only if $\texttt{dbMask} = \mathsf{MGF}(H, lenDB)$ agrees with $\texttt{dbMask}'_a$ on all but the leading p bits; this happens with probability $p_{3a} = 2^{-8lenDB+p} < 2^{-8hLen}$. To see why the inequality follows, recall that we set the parameters so that $emLen \geq \max(2hLen, hLen + sLen) + 2$; also recall that $lenDB = emLen - hLen - 1$ and $p \leq 7$.

Case 3b: a is not unique. Since $q^{\mathcal{A}}_{\mathsf{Hash}} \leq t_{\mathcal{A}}(k)$, the probability of this case is at most $\binom{q^{\mathcal{A}}_{\mathsf{Hash}}}{2} 2^{-8hLen} = \Theta(t^2_{\mathcal{A}}(k) 2^{-8hLen})$.

The above cases cover each situation in which event F may occur. By the union bound, $\Pr[F] \leq p_1 + p_2 + p_{3a} + p_{3b} = \Theta(t^2_{\mathcal{A}}(k)2^{-8hLen})$.

D.2 Proof of Claim 8

Let us compare $\mathcal{A}$'s view when interacting with the strong one-more-unforgeability challenger to its view when interacting with $\mathcal{B}$. Recall that RSA is random-self reducible, i.e., for every y, $w = yr^e$ sampled by first picking $r \leftarrow \mathbb{Z}_N$ and then computing w from it is distributed identically to w sampled uniformly at random from $\mathbb{Z}_N$.

Input to $\mathcal{A}$. In both experiments, $\mathcal{A}$ receives an RSA public key (N, e) as input, together with the other system parameters. Since the RSA modulus is generated using the same algorithm in both experiments, the input to $\mathcal{A}$ is distributed identically in the two experiments.

$\mathcal{A}$'s interaction with Hash. In the experiment with the challenger, $\mathsf{Hash}(\cdot, \ell)$ is a random oracle for every ℓ. $\mathcal{B}$'s responses to these queries will be truly random as well; let us ascertain that via case analysis:

- If $\mathcal{A}$ repeats a previous query, $\mathcal{B}$'s response is the same as it gave previously, which is identical to how a random oracle would respond to a repeat query.
- Recall that a relevant query to $\mathsf{Hash}(\cdot, \cdot)$ is a pair (v, ℓ) such that $\ell = hLen$ and the first eight bytes of v are all 0. If $\mathcal{A}$'s query is new and not relevant, then $\mathcal{B}$'s response to it is a truly random string of ℓ octets, identical to the response of a random oracle.
- If $\mathcal{A}$'s query is the i^{th} new and relevant query v_i, then $\mathcal{B}$ responds to it by first running a procedure that outputs a random $w_i \in \mathbb{Z}_N$ that starts with the requisite number of 0's, and then parsing $\mathtt{EM}_i = \mathsf{I2OSP}(w_i, emLen)$ as $\mathtt{EM}_i = \mathtt{maskedDB}_i \circ H_i \circ 0xBC$. Next, $\mathcal{B}$ fails if $\mathsf{MGF}(H_i, lenDB)$ is already defined; if $\mathcal{B}$ does not fail, then it outputs H_i. Since H_i is a substring of a freshly sampled string, it is a truly random string of the correct length. Therefore, in the event that $\mathcal{B}$ does not fail, the response to $\mathcal{A}$'s query is identical to that produced by a random oracle.

$\mathcal{A}$'s interaction with MGF. In the experiment with the challenger, $\mathsf{MGF}(\cdot, \ell)$ is a random oracle for every ℓ. $\mathcal{B}$'s responses to these queries will be truly random as well; let us ascertain that via case analysis:

- If $\mathsf{MGF}(u, \ell)$ has not been previously defined, then $\mathcal{B}$'s response is a truly random string of ℓ octets, identical to the output of a random oracle.
- If $\mathsf{MGF}(u, \ell)$ was previously defined by another query to MGF, then $\mathcal{B}$'s response is the same as its previous response to this query, identical to the output of a random oracle.
- If $\mathsf{MGF}(u, \ell)$ was previously defined when $\mathcal{A}$ called $\mathsf{Hash}(v_i, hLen)$, then $\mathcal{B}$'s response is $\mathtt{dbMask}_i$; note that it is the result of masking a string with an unused substring of the random w_i. Therefore, the response is identical to that produced by a random oracle.

$\mathcal{A}$'s interaction with $\mathsf{BSig}(SK,\cdot)$. In the experiment with the one-more-unforgeability challenger, $\mathcal{A}$ (essentially, ignoring the integer-to-string and vice versa conversions) sends to BSig values $m \in \mathbb{Z}_N$ and, in response, receives the values $s \in \mathbb{Z}_N$ such that $s^e = m \bmod N$. In the experiment with $\mathcal{B}$, $\mathcal{B}$ furnishes $\mathcal{A}$ with identical responses by using its inversion oracle.

As a result of the above side-by-side comparison, it is clear that, in the event that $\mathcal{B}$ does not fail, $\mathcal{A}$'s view when interacting with $\mathcal{B}$ is identical to its view when interacting with the challenger in the random-oracle model. To complete the proof of the claim, let us analyze the probability that $\mathcal{B}$ fails.

For $1 \leq i \leq q^{\mathcal{A}}_{\mathsf{Hash}}(PK;R)$, let E_i be the event that $\mathsf{MGF}(H_i, lenDB)$ is already defined during the response to $\mathsf{Hash}(v_i, hLen)$. Since the only way for $\mathcal{B}$ to fail while interacting with $\mathcal{A}$ is if some E_i happens, the event that $\mathcal{B}$ fails is $E = \bigcup_{i=1}^{q^{\mathcal{A}}_{\mathsf{Hash}}} E_i$. Let $a_1,\ldots,a_{q_i}$ be values such that $\mathsf{MGF}(a_j, lenDB)$ is already defined during the call to $\mathsf{Hash}(v_i, hLen)$; note that $q_i \leq q^{\mathcal{A}}_{\mathsf{Hash}} + q^{\mathcal{A}}_{\mathsf{MGF}}$ since just one value $\mathsf{MGF}(a, lenDB)$ is defined per oracle call to either Hash or MGF, and $q^{\mathcal{A}}_{\mathsf{Hash}}$ and $q^{\mathcal{A}}_{\mathsf{MGF}}$ are upper bounds on how many times the calls to these respective oracles are made. Since H_i is a $8hLen$-bit value picked uniformly at random at this stage (as we argued above), it collides with a_j with probability exactly $2^{-8hBLen}$; thus, by the union bound, $\Pr[E_i] \leq (q^{\mathcal{A}}_{\mathsf{Hash}} + q^{\mathcal{A}}_{\mathsf{MGF}})2^{-8hBLen}$.

Recall that $\mathcal{B}$ fails only if it fails while responding to $\mathsf{Hash}(v_i, hLen)$ for some i. Thus, by the union bound,

$$\begin{aligned}\Pr[E] &\leq \sum_{i=1}^{q^{\mathcal{A}}_{\mathsf{Hash}}} \Pr[E_i] \\ &\leq q^{\mathcal{A}}_{\mathsf{Hash}}(q^{\mathcal{A}}_{\mathsf{Hash}} + q^{\mathcal{A}}_{\mathsf{MGF}})2^{-8hLen} \\ &= O(t^2(k)2^{-8hLen})\end{aligned}$$

where the last equality follows because $q^{\mathcal{A}}_{\mathsf{Hash}} + q^{\mathcal{A}}_{\mathsf{MGF}} \leq t(k)$.

D.3 Proof of Claim 9

As we have already argued, we may assume without loss of generality that $\mathcal{A}$ is high-achieving, i.e. its output is either empty or consists of exactly $q^{\mathcal{A}}_{\mathsf{BSig}} + 1$ message-signature pairs that pass verification, where recall that $q^{\mathcal{A}}_{\mathsf{BSig}}$ is the number of signing queries made by $\mathcal{A}$.

By Claim 7, the probability that $\mathcal{A}$ succeeds and asks all the queries needed for verifying its output is at least $p_{\mathcal{A}}(k) - \Theta(t^2_{\mathcal{A}}(k)2^{-8hLen})$.

By Claim 8, conditioned on $\mathcal{B}$ not failing while responding to a query from $\mathcal{A}$, $\mathcal{A}$'s view when interacting with $\mathcal{B}$ is identical to its view in the one-more-unforgeability experiment. Therefore, conditioned on $\mathcal{B}$ not failing while responding to a query, $\mathcal{A}$ succeeds and asks all the queries needed for verifying its output with probability at least $p_{\mathcal{A}}(k) - \Theta(t^2_{\mathcal{A}}(k)2^{-8hLen})$. By Claim 8, $\mathcal{B}$ fails while responding to a query from $\mathcal{A}$ with probability $O(t^2_{\mathcal{A}}(k)2^{-8hLen})$.

Putting it together using the union bound: with probability at least $p_{\mathcal{A}}(k) - \Theta(t^2_{\mathcal{A}}(k)2^{-8hLen})$, it is the case that, when interacting with the reduction $\mathcal{B}$, all three of the following hold: (1) $\mathcal{B}$ has not failed while answering a query from $\mathcal{A}$; (2) $\mathcal{A}$ succeeds in outputting $q^{\mathcal{A}}_{\mathsf{BSig}} + 1$ valid and distinct message-signature pairs $\{(M_j, \sigma_j)\}$; (3) $\mathcal{A}$'s computation history includes all the queries to Hash and MGF that $\mathsf{PSSVerify}(PK, M_j, \sigma_j)$ makes for all $1 \leq i \leq q+1$.

Let us show that, in this case, $\mathcal{B}$ succeeds in solving the the RSA-ACTI problem.

Let (M, σ) be one of the message-signature pairs output by $\mathcal{A}$ conditioned on (1), (2) and (3) above. By (2), we know that it is accepted by the verification algorithm. Let $\mathtt{EM} = \mathtt{maskedDB} \circ H \circ 0xBC$, $\mathtt{dbMask}$, $\mathtt{DB}$, $\mathtt{salt}$ and M' be the values derived when verifying (σ, M). By (3), we know that $\mathcal{A}$ queried $\mathsf{Hash}(M', hLen)$ during its computation; we know that this was a relevant query from how M' is computed by the verification algorithm, so for some i, this was the relevant query $(v_i, hLen)$. Let $w_i = y_i r_i^e$, $\mathtt{EM}_i = \mathtt{maskedDB}_i \circ H_i \circ 0xBC$, $\mathtt{dbMask}_i$, $\mathtt{DB}_i$, $\mathtt{salt}_i$ be the values $\mathcal{B}$ defined when responding to query $(v_i, hLen)$.

Based on this observation, let us work through the steps of the verification algorithm and make a series of further observations:

According to the verification algorithm, $\mathtt{EM} = \mathsf{I2OSP}(\sigma^e \bmod N, emLen)$. The verification algorithm parses $\mathtt{EM} = \mathtt{maskedDB} \circ H \circ 0xBC$ (using the proper encoding lengths). Since we know that the verification algorithm ultimately accepted, we know that $0xBC$ is in fact the last byte of $\mathtt{EM}$. Moreover, since we know that the verification algorithm queried Hash on input v_i and received and accepted H_i, we know that $H = H_i$.

Recall that the verification algorithm computes $\mathtt{dbMask} = \mathsf{MGF}(H, lenDB)$. Since $H = H_i$, $\mathtt{dbMask} = \mathsf{MGF}(H_i, lenDB) = \mathtt{dbMask}_i$. Let $\mathtt{DB} = \mathtt{maskedDB} \oplus \mathtt{dbMask} = \mathtt{maskedDB} \oplus \mathtt{dbMask}'_i$, where recall that $\mathtt{dbMask}'_i$ is the same as $\mathtt{dbMask}_i$ except that the first p bits are zeroes.

If $\mathtt{DB}$ did not start with $a = 8(lenDB - sLen - 1)$ zero bits followed by $0x01$, then verification would reject. Since we know it accepted, we can parse $\mathtt{DB} = 0^a \circ 0x1 \circ \mathtt{salt}$ to obtain $\mathtt{salt}$. We know that $M' = 0^{64} \circ \mathtt{mHash} \circ \mathtt{salt} = v_i$, where $\mathtt{mHash} = \mathsf{CRHF}(M)$. Therefore, $\mathtt{salt} = \mathtt{salt}_i$. Therefore, $\mathtt{DB} = 0^{64} \circ 0x1 \circ \mathtt{salt} = 0^{64} \circ 0x1 \circ \mathtt{salt}_i = \mathtt{DB}_i$.

Putting everything together, $\mathtt{maskedDB} = \mathtt{DB} \oplus \mathtt{dbMask}'_i = \mathtt{DB}_i \oplus \mathtt{dbMask}'_i = \mathtt{maskedDB}_i$, and so $\mathsf{I2OSP}(\sigma^e \bmod N, emLen) = \mathtt{EM} = \mathtt{maskedDB} \circ H \circ 0xBC = \mathtt{maskedDB}_i \circ H_i \circ 0xBC = \mathtt{EM}_i = \mathsf{I2OSP}(w_i, emLen)$. Therefore, $\sigma^e = w_i = y_i r_i^e$, thus $(\sigma / r_i)^e = y_i$.

It remains to show that each of the $q^{\mathcal{A}}_{\mathsf{BSig}} + 1$ valid distinct message-signature pairs produced by $\mathcal{A}$ corresponds to a unique challenge y_i. Suppose (M_i, σ_i) and (M_j, σ_j) are two message-signature pairs output by $\mathcal{A}$ that correspond to the same relevant query v. Then $v = 0^{64} \circ \mathtt{mHash}_i \circ \mathtt{salt}_i = 0^{64} \circ \mathtt{mHash}_j \circ \mathtt{salt}_j$.

Suppose we have some $M_i \neq M_j$. Therefore, $\mathsf{Hash}(M_i, hLen) = \mathtt{mHash}_i = \mathtt{mHash}_j = \mathsf{Hash}(M_j, hLen)$ is a collision in the random oracle Hash. The probability that this may happen for some is at most $\binom{q^{\mathcal{A}}_{\mathsf{Hash}}}{2} 2^{-8hLen} = \Theta(t^2_{\mathcal{A}}(k) 2^{-8hLen})$.

Suppose that $M_i = M_j$. Since they correspond to the same relevant query v, we know that $\mathtt{salt}_i = \mathtt{salt}_j$. Since the signature is fully determined once the message and the salt are fixed, it follows that $\sigma_i = \sigma_j$, and so $(M_i, \sigma_i) = (M_j, \sigma_j)$, therefore these two message-signature pairs are not distinct, thus they cannot be output by a high-achieving $\mathcal{A}$.

Thus, with probability at least $p_{\mathcal{A}}(k) - \Theta(t^2_{\mathcal{A}}(k)2^{-8hLen})$, $\mathcal{B}$ has computed $q^{\mathcal{A}}_{\mathsf{BSig}}+1$ solutions to the challenges $y_1, \ldots, y_{q^{\mathcal{A}}_{\mathsf{Hash}}}$ from $\mathcal{B}$'s challenge oracle $\mathcal{O}_R(\cdot, N)$. Therefore, $\mathcal{B}$ wins the RSA-ACTI game with probability at least $p_{\mathcal{A}}(k) - \Theta(t^2_{\mathcal{A}}(k)2^{-8hLen})$.