The Security of Practical Two-Party RSA Signature Schemes

Mihir Bellare[*] Ravi Sandhu[†]

Abstract

In a two-party RSA signature scheme, a client and server, each holding a share of an RSA decryption exponent d, collaborate to compute an RSA signature under the corresponding public key N, e known to both. This primitive is of growing interest in the domain of server-aided password-based security, where the client's share of d is based on its password. To minimize cost, designers are looking at very simple, practical protocols based on the early ideas of Boyd, but their security is unclear. We analyze a class of these protocols. We suggest two notions of security for two-party signature schemes and provide proofs of security for the schemes in our class based on assumptions about RSA and the hash function underlying the scheme.

Keywords: RSA, digital signatures, two-party protocols, proofs of security.

[*]Dept. of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA, and SingleSignOn.net. E-Mail: `mihir@cs.ucsd.edu`. URL: `http://www-cse.ucsd.edu/users/mihir`.

[†]ISE Department, Mail Stop 4A4, George Mason University, Fairfax, VA 22030-4444, USA, and SingleSignOn.net. E-Mail: `rsandhu@singlesignon.net`. URL: `http://www.list.gmu.edu/~sandhu/`.

Contents

1 Introduction

We consider a standard RSA-based signature scheme of the "hash-then-decrypt" variety, meaning the public key is N, e, the secret key is d, and the signature of message M is $H(M)^d \bmod N$, where H is a public hash function, N is an RSA modulus, e is an encryption exponent, and d is the corresponding decryption exponent. However, instead of there being a single signer, the public key is associated to a pair of entities that we call the client and the server. The decryption exponent d is not held by any individual party, but rather is split into shares d_c and d_s, and these are held by the client and server, respectively. A collaborative computation, or signing protocol, is used to produce a signature for the client.

Basic idea. RSA, due to its algebraic properties, lends itself naturally to collaborative signature computation. The earliest suggestion of which we are aware is due to Boyd [5]. The decryption exponent is split multiplicatively, meaning

$$d_c d_s \equiv d \pmod{\varphi(N)}\ . \tag{1}$$

Collaborative signature computation is then based on the equation

$$H(M)^d \equiv H(M)^{d_c d_s} \bmod N\ . \tag{2}$$

While this basic idea is old and well-known, it does not seem to have been appreciated that variations in the way it is used give rise to different protocols that have perhaps unexpectedly different security properties.

The protocols. This paper considers a family of natural and simple two-party signature schemes based on direct exploitation of Equation (2). We divide them into two classes. In the *common-message* class of schemes, the message M to be signed is a common input to client and server. Within this class we consider two protocols, differing according to who goes first:

MCS : Client sends $x_c = H(M)^{d_c} \bmod N$ to the server; server computes signature $x = x_c^{d_s} \bmod N$, verifies it, and returns it to client.

MSC : Server sends $x_s = H(M)^{d_s} \bmod N$ to client; client computes signature $x = x_s^{d_c} \bmod N$.

In the *client-message* class of schemes, the message M to be signed is input to the client but not to the server. Within this class we again consider two protocols, differing according to who goes first with regard to computing a "partial" signature:

HCS : Client sends y, x_c to the server where $y = H(M)$ and $x_c = y^{d_c} \bmod N$; server computes signature $x = x_c^{d_s} \bmod N$, verifies that $x^e \equiv y \pmod{N}$, and returns x to client.

HSC : Client sends y to the server where $y = H(M)$; server sends $x_s = y^{d_s} \bmod N$ to client; client computes signature $x = x_s^{d_c} \bmod N$.

The leading "M" in the names of the common-message protocols stands for the "Message" that both parties know, while the leading "H" in the names of the client-message protocols stands for the "Hash" that the client flows to the server. The other letters reflect the order in which the Client and Server use their shares of the decryption exponent in the protocol.

In this paper we analyze the security of the above four protocols with regard to meeting well-defined modern cryptographic goals in provable ways. We find that the security goals, and the assumptions on the underlying primitives required to prove security, vary from protocol to protocol in a perhaps surprising way.

Motivation. This work is motivated by recent practical interest in two-party RSA signature schemes. Several companies are working on products that enable server-aided password-based cryptography. Ganesan [13] had suggested that d_c be based on a client password. The server could be implemented via a smartcard, or be a device on the network as in [26]. This is an attractive model in the face of the difficulties of PKI, especially because of the so-called "instant revocation" capability it offers. Upon password compromise, the client can revoke the public key by informing the server that it is no longer valid. Since every signature produced under the public key involves the server, revocation takes immediate effect, unlike in PKI where there may be a time-lag between actual revocation and a signature-verifier's receipt of the revocation list.

Why all these protocols? In developing protocols based on Equation (2), practitioners are likely to explore variations based on benefits they bring in terms of functionality, performance, and ease of implementation. The four protocols above represent some lines along which choices can be made, as we now discuss.

In providing the message M to be signed to both parties, the common-message protocols represent the abstraction, standard in secure two-party computation (cf. [27]), that relevant inputs are agreed upon in advance. These are the first protocols to consider, and they deserve analysis. In practice however, the message typically originates with the client, who must flow it to the server if the latter is to be made aware of it. Practioners may want to avoid flowing the message, instead flowing the short message hash. (Motivations vary. If the server is a smartcard, it will not have storage for the long message. Even if the server is on the network, one prefers to reduce bandwidth by sending the short message-hash rather than the possibly long message. In either setting, a client may prefer to avoid the blatant violation of privacy represented by sending the message to be signed to the server.) This leads us to consider the client-message class of protocols in this paper. The other dimension, namely whether the client or the server is the first to compute a partial signature, is a natural one, and implementers may have a preference based on some performance or convenience criterion. This leads us to consider two protocols in each class.

Implementers or would be implementers who are choosing between these protocols, based on performance or functionality, would benefit from being aware of how they compare with regard to security. The analyses and results in this paper are directed at providing such information.

Security goals. The first question to ask towards providing meaningful security analyses is: what are the adversarial models and target security goals for these protocols?

In line with the types of applications being envisaged, we view the server as trusted. The first security goal that comes to mind is to prevent forgery by a third party. We suggest however that this goal is too weak, and instead ask that forgery be hard even for an adversarial client who is in possession of the correct share d_c and is allowed to engage in interactions with the d_s-holding server. (Security against third parties is implied by security against client adversaries, so consideration of the latter only strengthens the security results.) This is appropriate because we view the server is a "co-signer" of the client. A verifier who accepts a client signature does so under the belief that the server "endorses" it, so a client who succeeds in creating a signature that the server has not endorsed should be viewed as having been successful in forgery.

When we seek to formulate the goals more precisely, a dichotomy emerges depending on the class of protocol. In the case of common-message schemes, we formulate a notion of security against forgery under chosen-message attack analogous to that for standard schemes [19]. (It asks that the client-adversary should find it hard to output a valid signature of a message that it did not previously provide as input to a session with the server.) Client-message schemes, however, are more like blind signature schemes than standard ones, in that there need be no clearly identifiable message associated to an interaction between a client-adversary and the server. Accordingly we formulate a notion of security against one-more-forgery, analogous to that for blind signature schemes [22]. (It asks that the client-adversary should find it hard to output a number of valid message-signature pairs that exceeds the number of sessions of interaction in which it engaged the server.) Ours is the first work to consider client-message schemes, and we consider this, together with the introduction of a notion of security for them, to be one of the contributions of this paper.

Implementers should realize that two-party signature schemes from the two different classes are inherently different with regard to the type of security they can achieve, so that they can think about which goal they want to target. However, one should not view the difference as a negative connotation with regard to client-message schemes. The goal of security against one-more forgery that they target is useful and in line with the idea that the role of the server is to "endorse" client messages: the verifier's belief on accepting a signature is not that the server stands behind the message, but that it authorized a signature, on a message of the client's choice, at this point in time.

Results. Our results are represented in Figure 1. The middle column of Figure 1 lists assumptions about RSA and the hash function H, and the strength of the assumptions in this column decreases as one moves down. (The RSA-related computational problems RSA-STI, RSA-CTI and Split-Key-RSA-CTI are discussed in Section 4.) The security statements regarding our four schemes are in the four corner boxes. The arrows show under what assumptions we can establish the security claims in question. The results and their implications are discussed at length in Section 5. Proofs of the theorems stated there are in Section 6.

Background and related work. Boyd's suggestion [5] was to split the decryption exponent as per Equation (1) and then sign message $m \in \mathsf{Z}_N^*$ by exploiting the equation $m^d \equiv m^{d_c d_s} \bmod N$. Security analysis of the above was initiated in [5] and continued in [14, 15], but pertains to relatively weak security goals such as secret-key-recovery since the "plain" RSA signatures that these schemes create are well-known to be subject to forgery attack. To obtain usable protocols and analyses targetting strong notions of security, we have explicitly considered and modeled the hash function.

One can consider splitting the decryption exponent additively rather than multiplicatively, namely the shares d_c, d_s satisfy $d_c + d_s \equiv d \pmod{\varphi(N)}$. Collaborative signature computation is then based on the equation $H(M)^d \equiv H(M)^{d_c} \cdot H(M)^{d_s} \pmod{N}$. The issues and analyses for this case are similar to, and somewhat simpler than, those for the case of multiplicative splitting that we consider, and in order to avoid repetitiveness, we do not detail the results related to additive splitting.

MacKenzie and Reiter [20] consider a setting where the client is implemented as a password-controlled device. The core of their S-RSA protocol is the additive-splitting version of MSC. This is enhanced to achieve numerous security properties that we do not consider here. (Their device does not hold d_c but rather holds an encryption of it under the server public key, so that compromise of the device does not provide d_c to the adversary.) Their proof of security for S-RSA includes at its core a proof of security against forgery under chosen-message attack for the additive-splitting version of MSC. Since adapting this to the multiplcative-splitting based MSC we consider here is entirely direct, the result about the security of MSC that we state as Theorem 5.5 should be considered as due to [20]. (We provide a proof for completeness and because it is a useful introduction to the other proofs in this paper, but the ideas in the proof are entirely those of [20].) We remark that our consideration of the HCS and HSC protocols reflects some questions raised in [20] regarding the loss in privacy incurred by MSC in providing the server with the message to be signed.

Two-party signature schemes are part of the general approach of secure two-party computation [27] and threshold cryptography [11]. Threshold cryptography has however concentrated more on the case of $n \geq 3$ parties. Distribution of the RSA function (which yields distributed signatures) is considered in this setting in [11, 10, 17]. Split-key schemes for DSA signatures are proposed in [21].

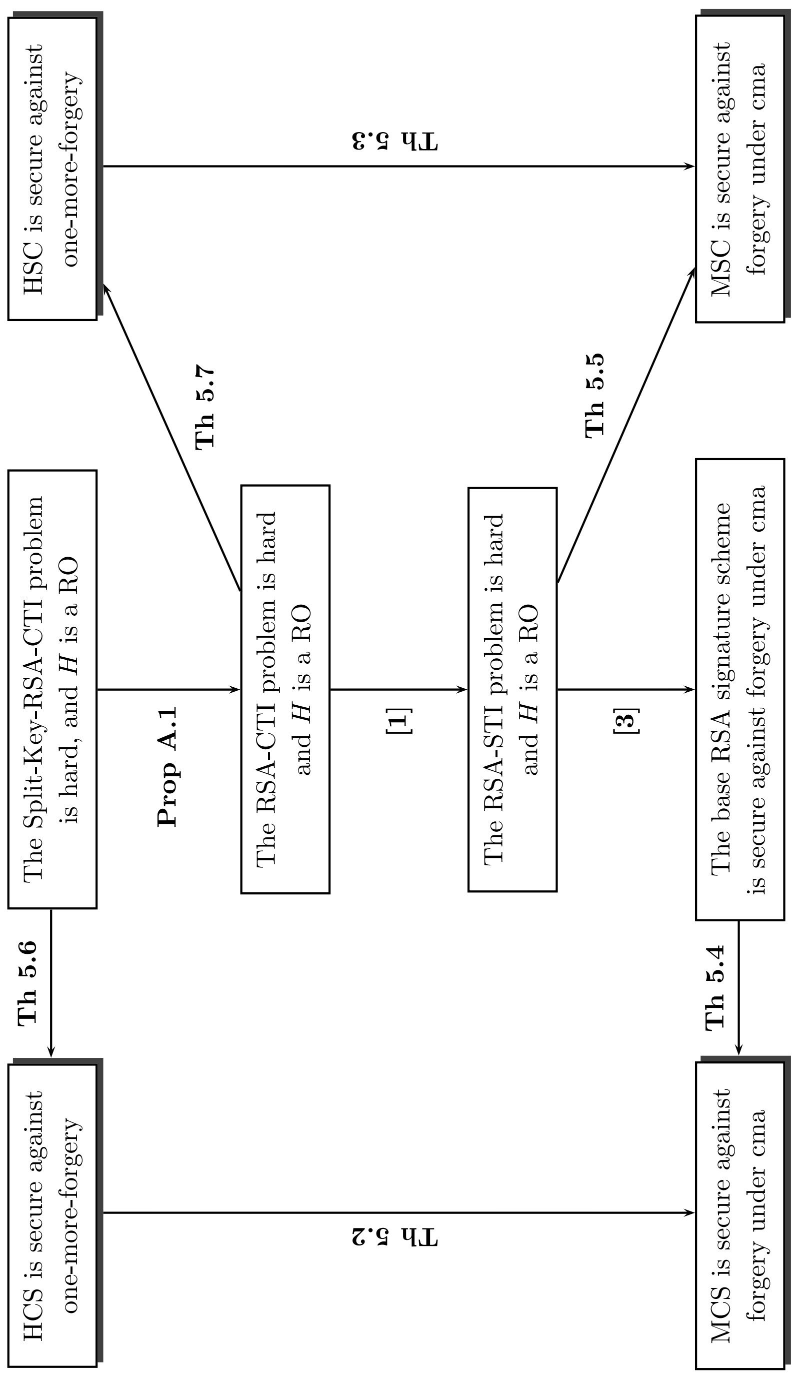
HSC is secure against one-more-forgery
Th 5.3
MSC is secure against forgery under cma
Th 5.7
Th 5.5
The Split-Key-RSA-CTI problem is hard, and H is a RO
Prop A.1
The RSA-CTI problem is hard and H is a RO
[1]
The RSA-STI problem is hard and H is a RO
[3]
The base RSA signature scheme is secure against forgery under cma
Th 5.6
Th 5.4
HCS is secure against one-more-forgery
Th 5.2
MCS is secure against forgery under cma

Figure 1: **Results:** Here $\mathbf{A} \rightarrow \mathbf{B}$ means that if **A** is true then so is **B**. That is, if we assume **A** then we can prove **B**. The boxes in the middle column represent assumptions about RSA and the hash function H, and the strength of the assumptions decreases as we move down. The four corners of the picture indicate our four two-party signature schemes with their associated security goals. Arrows are annotated with pointers to justifications, in some cases a Theorem or Proposition in this paper, in other cases a known result for which a citation is provided.

2 The setup and the schemes

RSA KEY GENERATORS. An *RSA modulus* is an integer which is a product of two distinct, odd primes. An *RSA key generator* is a (randomized) algorithm which on input of a security parameter k returns N, e, d, p, q where p, q are primes, $N = pq$ is an RSA modulus satisfying $2^{k-1} \leq N < 2^k$, and $e, d \in \mathsf{Z}^*_{\varphi(N)}$ are, respectively, encryption and decryption exponents, satisfying $ed \equiv 1 \pmod{\varphi(N)}$. (Here $\varphi(\cdot)$ is the Euler totient function.)

There are many different possible RSA key generators, varying for example in the choice of e or the types of primes chosen. In order to highlight the fact that the definitions and results here apply to any RSA key generator, we do not pin down any particular generator, but instead parameterize definitions and results by a choice of generator.

BASE SIGNATURE SCHEME. We associate to any RSA key generator KG a *base digital signature scheme* DS = (KG, SIGN, VERIFY) of the standard "hash-then-decrypt" variety. The client has public key (N, e) and secret key (N, d), where N, e, d are obtained by running KG(k). The client's signature of a message M is the value $\text{SIGN}^H_{N,d}(M) = H(M)^d \bmod N$, where $H\colon \{0,1\}^* \to \mathsf{Z}^*_N$ is a public hash function, and verification is defined via $\text{VERIFY}^H_{N,e}(M, x) = 1$ iff $x^e \equiv H(M) \pmod{N}$. The bulk of standardized schemes have this form, although they might differ in how they implement the hash function H (cf. [24]).

THE TWO-PARTY SETTING. In the stand-alone client setting which one usually envisages, the client is in possession of the decryption exponent d that enables signature computation. In the two-party setting, the client does not hold d. Instead, d is "split" into two parts, d_c and d_s, called the *shares* of d. The client holds d_c while the server holds d_s. (Additionally, both parties know N, e). When the client wishes to obtain the signature $x = H(M)^d \bmod N$ of a message M, it interacts with the server to this end, via a *two-party signature protocol*.

There are several ways in which d might be split. In an *additive split*, $d_c + d_s \equiv d \pmod{\varphi(N)}$, while in a *multiplicative split*, $d_c d_s \equiv d \pmod{\varphi(N)}$. Either type of split lends itself equally well to two-party based signing (cf. [5]). In this paper we will focus on the multiplicative split.

We require *compatibility* with the underlying base signature scheme, in the sense that the signature computed as a result of the signature protocol is required to be the same as would have been computed in the stand-alone setting. In other words, although the signature generation process has changed, the signature verification algorithm has not.

TWO-PARTY SIGNATURE SCHEMES. Let KG be an RSA key generation algorithm and let DS be an associated base signature scheme. The specification of a particular DS-*compatible two-party signature scheme* takes the form SDS = ($\overline{\text{KG}}$, CLIENT, SERVER, VERIFY), where the last component VERIFY is the verification algorithm of the base signature scheme, and the other components are as follows.

The *split-key generation algorithm* $\overline{\text{KG}}$, on input k, first runs KG to obtain N, e, d, p, q, and then splits d into the shares d_c, d_s, returning an output N, e, d_c for the client, and N, e, d_s for the server. In this paper, the splitting process is fixed to the following: the algorithm chooses d_c at random from $\mathsf{Z}^*_{\varphi(N)}$ and then sets $d_s = d \cdot d_c^{-1} \bmod \varphi(N)$. (Notice that in order to do this, the algorithm must know $\varphi(N)$, which it does since the latter equals $(p-1)(q-1)$, and it has p, q via the output of KG.) Thus $\overline{\text{KG}}$ is fully specified once KG is specified.

We imagine that the split-key generation algorithm is run by a trusted dealer who securely delivers each party its share, and then dies. (In practice, there are many ways that one might chose to perform the splitting of d and distribution of shares. One possibility is a key generation protocol, either between the two parties [23, 18] or involving another party [4, 12]. Another possibility is that it is done by a trusted party who in some cases might be the server itself. We are not concerned here with this issue, but rather with the security of the signature protocols, and accordingly assume a magically and correctly performed distribution of the shares.)

The Client and Server components of the scheme represent the code that the client and server, respectively, are supposed to execute in the protocol to obtain a joint signature of a message. The client code is initialized with N, e, d_c, while the server code is initialized with N, e, d_s. The client code additionally takes input a message M, and then initiates an interaction with the server. At the end of the interaction the client outputs a value $x \in \mathsf{Z}_N^*$. It is required that an interaction between properly initialized Client and Server on server input M result in output of the signature of M, meaning $\text{Verify}_{N,e}^H(M, x) = 1$.

Types of two-party schemes. We distinguish two types of two-party signature schemes. In a *common-message* scheme, the message M to be signed is input not only to the client but also to the server. (There are various ways in which this might happen in practice, the most common of which is that the client flows M to the server. However, as our goal again is to be general, we do not wish to make an assumption regarding how the server knows the message, and in the model simply provide it as an input to the server. We stress that it is a model assumption that the two parties have the *same* message as input, and also that the Server code expects M as input and will not function if this is not provided.) In a *client-message* scheme, the message M to be signed is provided as input to the client but not to the server. (This does not mean that the server does *not* know M, since it might as a result of the protocol flows. But we do not assume the server knows M.) This type of scheme reflects the possibility that the client will prefer not to flow the message, either because it wants to keep the message private, or because it wants to save on bandwidth on server-storage by transmitting the short message-hash instead of the (possibly long) message.

As we will see later, the motivation for separating these two types of schemes is that security-wise, they cannot be treated equally: the notions of security they can target are necessarily different.

Four two-party signature schemes. In this paper we consider the four specific two-party signature schemes whose signing protocols are illustrated in Figure 2. (The schemes all have the same split-key generation algorithm $\overline{\text{KG}}$ and verification algorithm Verify, as dictated by the underlying base signature scheme with which they are compatible, so we tend to identify each scheme with the description of its Client and Server components as given in the figures). The schemes in the first row of Figure 2 are of the common-message type, while those in the second row are of the client-message type. In this paper we are interested in the security of these four two-party signature schemes, with particular regard to answering the following question about each scheme: What can be proven about the security of the scheme, and under what assumptions on the underlying primitives?

3 Security notions for two-party signature schemes

Security of the base signature scheme. The security requirement of a standard signature scheme (in our case, the base signature scheme DS) is that it be secure against forgery under chosen-message attack [19]. This is formalized by associating to any Fg algorithm and value k of the security parameter an experiment, as follows. The forger is given input the public key N, e and has oracle access to $\text{Sign}_{N,d}^H(\cdot)$, the keys being selected via $\text{KG}(k)$. (If we are in a random oracle model, it also has oracle access to H, the latter chosen at random [2].) The forger is said to be *successful* if it outputs a pair M, x such that $\text{Verify}_{N,e}^H(M, x) = 1$ but M was not "legitimately signed," meaning a query to the sign oracle. The *advantage* of Fg, denoted $\mathbf{Adv}_{\text{DS,FG}}^{\text{uf-cma}}(k)$, is its

MCS

CLIENT		SERVER
Input: N, e, d_c, M		Input: N, e, d_s, M
$x_c \leftarrow H(M)^{d_c} \bmod N$		
	$\xrightarrow{x_c}$	
		$x \leftarrow x_c^{d_s} \bmod N$
		If $x^e \not\equiv H(M) \pmod N$
		then $x \leftarrow \perp$
	$\xleftarrow{x}$	
Output: x		

MSC

CLIENT		SERVER
Input: N, e, d_c, M		Input: N, e, d_s, M
		$x_s \leftarrow H(M)^{d_s} \bmod N$
	$\xleftarrow{x_s}$	
$x \leftarrow x_s^{d_c} \bmod N$		
Output: x		

HCS

CLIENT		SERVER
Input: N, e, d_c, M		Input: N, e, d_s
$y \leftarrow H(M)$		
$x_c \leftarrow y^{d_c} \bmod N$		
	$\xrightarrow{y, x_c}$	
		$x \leftarrow x_c^{d_s} \bmod N$
		If $x^e \not\equiv y \pmod N$
		then $x \leftarrow \perp$
	$\xleftarrow{x}$	
Output: x		

HSC

CLIENT		SERVER
Input: N, e, d_c, M		Input: N, e, d_s
$y \leftarrow H(M)$		
	$\xrightarrow{y}$	
		$x_s \leftarrow y^{d_s} \bmod N$
	$\xleftarrow{x_s}$	
$x \leftarrow x_s^{d_c} \bmod N$		
Output: x		

Figure 2: **Two-party RSA signing protocols.**

probability of success, taken over the coins of $\textsc{kg}(k)$, the coins of $\textsc{Fg}$ if any, and the random choice of H if we are in a random oracle model. The scheme $\textsc{ds}$ is said to be *secure against forgery under cma* if the function $\mathbf{Adv}^{\text{uf-cma}}_{\text{DS,FG}}(\cdot)$ is negligible for every poly(k)-time algorithm $\textsc{Fg}$.

Issues in security of two-party signature schemes. The security model we adopt is that the server is trusted, but the client is not. We imagine that the correct client code $\textsc{Client}$ is replaced by an adversary $\textsc{Bc}$. The latter is initialized with N, e and the client's share d_c of the decryption exponent, and will engage the (trusted) server $\textsc{Server}^H_{N,e,d_s}$ (the subscripts denoting the keys with which the server is initialized) in a sequence of interactions, with the purpose of eventually producing a forgery. (Note that this adversarial model is stronger than one in which we consider a "mere" forger who has N, e and oracle access to $\textsc{Client}^H_{N,e,d_c}$ and $\textsc{Server}^H_{N,e,d_s}$. We are allowing the forger to additionally "corrupt" the client and obtain d_c, so that our $\textsc{Bc}$ can certainly do anything that a "mere" forger can do, but possibly can do more.)

With this setup, we would like to mimic the notion of security for the base scheme. We would say that the adversary is successful if it outputs a pair M, x such that $\textsc{Verify}^H_{N,e}(M, x) = 1$ but M was "not legitimately signed." The difficulty is in how to interpret the phrase in quotes, and it here that we see the motivation for having made a distinction between common-message and client-message schemes. In a common-message scheme, each interaction with the server has a clearly identified message input for the server, to which the adversary must commit to start the interaction. In this case, "not legitimately signed" can be interpreted just as in the case of the base scheme, meaning $\textsc{Server}^H_{N,e,d_s}$ has not engaged in an interaction with the message in question as input. In a client-message scheme, however, interactions between an adversary client and $\textsc{Server}^H_{N,e,d_s}$ may not have any identifiable underlying message, so the phrase "not legitimately signed" has no reasonable interpretation. This situation is analogous to that in a blind signature scheme, and accordingly we will turn to the definitional approach of "one-more-forgery" used in that context [22]: success for the adversary means that it outputs a number of valid message-signature pairs that exceeds the number of interactions in which it engaged with the server. Let us now detail the two definitions.

Security of common-message schemes. Let $\textsc{sds} = (\overline{\textsc{kg}}, \textsc{Client}, \textsc{Server}, \textsc{Verify})$ be a two-party, common-message signature scheme. We associate to any adversary client $\textsc{Bc}$ and value k of the security parameter an experiment, as follows. Run $\overline{\textsc{kg}}(k)$ to get (N, e, d_c) and (N, e, d_s). (If we are in a random oracle model, also pick H at random.) Provide $\textsc{Bc}$ with input N, e, d_c and allow it to engage in interactions with $\textsc{Server}^H_{N,e,d_s}$. (If we are in a random oracle model, also give it oracle access to H.) Since the scheme is a common-message one, $\textsc{Bc}$ must begin each interaction with $\textsc{Server}^H_{N,e,d_s}$ by specifying a message to server as input to the server. However $\textsc{Bc}$ can choose client messages as it pleases in these interactions, and in particular differently from what $\textsc{Client}_{N,e,d_c}$. The adversary $\textsc{Bc}$ is said to be *successful* if it outputs a pair M, x such that $\textsc{Verify}^H_{N,e}(M, x) = 1$ but M was never provided by $\textsc{Bc}$ as message-input to $\textsc{Server}_{N,e,d_s}$. The *advantage* of $\textsc{Bc}$, denoted $\mathbf{Adv}^{\text{uf-cma}}_{\text{SDS,BC}}(k)$, is its probability of success, taken over the coins of $\overline{\textsc{kg}}(k)$, the coins $\textsc{Server}_{N,e,d_s}$, the coins of $\textsc{Bc}$ if any, and the random choice of H if we are in a random oracle model. The scheme $\textsc{sds}$ is said to be *secure against forgery under cma* if the function $\mathbf{Adv}^{\text{uf-cma}}_{\text{SDS,BC}}(\cdot)$ is negligible for every poly(k)-time client adversary $\textsc{Bc}$.

Security of client-message schemes. Let $\textsc{sds} = (\overline{\textsc{kg}}, \textsc{Client}, \textsc{Server}, \textsc{Verify})$ be a two-party, client-message signature scheme. We associate to any adversary client $\textsc{Bc}$ and value k of the security parameter an experiment, as follows. Run $\overline{\textsc{kg}}(k)$ to get (N, e, d_c) and (N, e, d_s). (If we are in a random oracle model, also pick H at random.) Provide $\textsc{Bc}$ with input N, e, d_c and allow it to engage in interactions with $\textsc{Server}^H_{N,e,d_s}$. (If we are in a random oracle model, also give it oracle access to H.) In these interactions, $\textsc{Bc}$ can choose client messages as it pleases, and in particular make choices different from those of $\textsc{Client}_{N,e,d_c}$. The adversary $\textsc{Bc}$ is said to be *successful* if it outputs a sequence of pairs $(M_1, x_1), \ldots, (M_{m+1}, x_{m+1})$ such that the following are true: (1) $\textsc{Verify}^H_{N,e}(M_i, x_i) = 1$ for all $i = 1, \ldots, m+1$, and (2) the number of interactions that $\textsc{Bc}$ initiated with $\textsc{Server}_{N,e,d_s}$ is at most m. The *advantage* of $\textsc{Bc}$, denoted $\mathbf{Adv}^{\mathrm{omf}}_{\mathrm{SDS,FG}}(k)$, is its probability of success, taken over the coins of $\overline{\mathrm{KG}}(k)$, the coins $\textsc{Server}_{N,e,d_s}$, the coins of $\textsc{Bc}$ if any, and the random choice of H if we are in a random oracle model. The scheme $\textsc{sds}$ is said to be *secure against one-more-forgery* if the function $\mathbf{Adv}^{\mathrm{omf}}_{\mathrm{SDS,Bc}}(\cdot)$ is negligible for every poly(k)-time client adversary $\textsc{Bc}$.

4 The RSA-related computational problems

In this section, we provide definitions of the three RSA related computational problems RSA-STI, RSA-CTI, and Split-Key-RSA-CTI on whose assumed security are results are based. We present and prove some relations between these problems in Appendix A. The definitions are parameterized by an arbitrary RSA key generator $\textsc{kg}$.

RSA-STI. Let $\textsc{kg}$ be an RSA key generator and let $\textsc{In}$ be an algorithm, referred to in this context as an rsa-sti-adversary. Consider the following experiment:

```
Experiment $\mathbf{Exp}^{\text{rsa-sti}}_{\textsc{kg},\,\textsc{In}}(k)$
    $(N, e, d, p, q) \stackrel{R}{\leftarrow} \textsc{kg}(k)$ ; $y \stackrel{R}{\leftarrow} \mathsf{Z}^*_N$ ; $x \leftarrow \textsc{In}(N, e, k, y)$
    If $x^e \equiv y \pmod N$ then return 1 else return 0
```

The *rsa-sti-advantage* of $\textsc{In}$, denoted $\mathbf{Adv}^{\text{rsa-sti}}_{\textsc{kg},\,\textsc{In}}(k)$, is the probability that experiment $\mathbf{Exp}^{\text{rsa-sti}}_{\textsc{kg},\,\textsc{In}}(k)$ returns 1. The RSA-STI problem is said to be *hard* with respect to $\textsc{kg}$ —in more standard terminology, RSA is said to be *one-way* with respect to $\textsc{kg}$— if the function $\mathbf{Adv}^{\text{rsa-sti}}_{\textsc{kg},\,\textsc{In}}(\cdot)$ is negligible for any poly(k)-time $\textsc{In}$. The assumption that RSA-STI is hard is simply the assumption that RSA is a one-way function, and is standard, but for consistency with problems considered later in this paper we use the phrase "RSA-STI is hard" in place of "RSA is a one-way function."

RSA-CTI. The RSA-CTI problem is one of a class of problems defined, and related to each other, in [1]. They showed that the assumption that the RSA-CTI problem is hard suffices to prove the security against one-more-forgery of Chaum's RSA-based blind signature scheme [7] in the random oracle model. The resemblance of the client-message two-party schemes to blind signature schemes with regard to security goals (one-more-forgery) makes it natural that the RSA-CTI problem is relevant here too. The problem itself is extension of the RSA-STI problem in which the adversary gets an inversion oracle, namely an oracle for $(\cdot)^d \bmod N$, and must invert RSA at $m+1$ target points using at most m oracle calls. The adversary must choose the $m+1$ target points from a sequence of $n(k)$ random input points. The function π reflects the choice it makes. We consider n as a parameter of the problem.

Let $\textsc{kg}$ be an RSA key generator and let $n : \mathsf{N} \to \mathsf{N}$ be a (polynomially bounded, polynomial time computable) function of the security parameter such that $n(k) \geq 1$ for all $k \in \mathsf{N}$. Let $\textsc{In}$ be an algorithm with access to an oracle, referred to in this context as an rsa-cti-adversary. Consider the following experiment:

```
Experiment $\mathbf{Exp}^{\text{rsa-cti}}_{\textsc{kg},\,\textsc{In},\,n}(k)$
    $(N, e, d, p, q) \stackrel{R}{\leftarrow} \textsc{kg}(k)$ ;  For $i = 1$ to $n(k)$ do $y_i \stackrel{R}{\leftarrow} \mathsf{Z}^*_N$
    $(m, \pi, x_1, \ldots, x_{m+1}) \leftarrow \textsc{In}^{(\cdot)^d \bmod N}(N, e, k, y_1, \ldots, y_{n(k)})$
    If the following are all true then return 1 else return 0
```

— $0 \leq m < n(k)$
— $\pi\colon \{1,\ldots,m+1\} \to \{1,\ldots,n(k)\}$ is an injective function
— $\forall i \in \{1,\ldots,m+1\} : x_i^e \equiv y_{\pi(i)} \pmod N$
— IN made at most m oracle queries

The *rsa-cti-advantage* of IN, denoted $\mathbf{Adv}^{\text{rsa-cti}}_{\text{KG},\,\text{IN},\,n}(k)$, is the probability that experiment $\mathbf{Exp}^{\text{rsa-cti}}_{\text{KG},\,\text{IN},\,n}(k)$ returns 1. The RSA-CTI$_n$ problem is said to be *hard* with respect to KG if the function $\mathbf{Adv}^{\text{rsa-cti}}_{\text{KG},\,\text{IN},\,n}(\cdot)$ is negligible for any poly(k)-time IN. The RSA-CTI problem is said to be hard with respect to KG if RSA-CTI$_n$ is hard for all polynomially-bounded, polynomial-time computable functions $n(\cdot) \geq 1$.

The RSA-CTI$_1$ problem (ie. the case $n(\cdot) = 1$) is identical to the RSA-STI problem, and it is in this sense that the hardness of the RSA-CTI problem can be considered an extension of the standard one-wayness assumption about RSA.

SPLIT-KEY-RSA-CTI. We introduce in this paper a Split-Key extension of the RSA-CTI problem. Let d_1, d_2 be shares of d, meaning $d_1, d_2 \in \mathsf{Z}^*_{\varphi(N)}$ and $d_1 d_2 \equiv d \pmod{\varphi(N)}$. The adversary has e, d_1 as input, and an oracle for $(\cdot)^{d_2} \bmod N$. The rest is as in RSA-CTI. Namely let KG be an RSA key generator and let $n : \mathsf{N} \to \mathsf{N}$ be a (polynomially bounded, polynomial time computable) function of the security parameter such that $n(k) \geq 1$ for all $k \in \mathsf{N}$. Let IN be an algorithm with access to an oracle, referred to in this context as an sk-rsa-cti adversary. Consider the following experiment:

```
Experiment $\mathbf{Exp}^{\text{sk-rsa-cti}}_{\text{KG},\,\text{IN},\,n}(k)$
    $(N,e,d,p,q) \xleftarrow{R} \text{KG}(k)$ ; $M \leftarrow (p-1)(q-1)$ ; $d_1 \xleftarrow{R} \mathsf{Z}^*_M$ ; $d_2 \leftarrow d \cdot d_1^{-1} \bmod M$
    For $i = 1$ to $n(k)$ do N $y_i \xleftarrow{R} \mathsf{Z}^*_N$ EndFor
    $(m, \pi, x_1, \ldots, x_{m+1}) \leftarrow \text{IN}^{(\cdot)^{d_2} \bmod N}(N, e, d_1, k, y_1, \ldots, y_{n(k)})$
    If the following are all true then return 1 else return 0
        — $0 \leq m < n(k)$ and $\pi\colon \{1,\ldots,m+1\} \to \{1,\ldots,n(k)\}$ is injective
        — $\forall i \in \{1,\ldots,m+1\} : x_i^e \equiv y_{\pi(i)} \pmod N$
        — IN made at most $m$ oracle queries
```

The *sk-rsa-cti-advantage* of IN, denoted $\mathbf{Adv}^{\text{sk-rsa-cti}}_{\text{KG},\,\text{IN},\,n}(k)$, is the probability that $\mathbf{Exp}^{\text{sk-rsa-cti}}_{\text{KG},\,\text{IN},\,n}(k)$ returns 1. The Split-Key-RSA-CTI$_n$ problem is said to be *hard* with respect to KG if the function $\mathbf{Adv}^{\text{sk-rsa-cti}}_{\text{KG},\,\text{IN},\,n}(\cdot)$ is negligible for any poly(k)-time IN. The Split-Key-RSA-CTI problem is said to be hard with respect to KG if Split-Key-RSA-CTI$_n$ is hard for all polynomially-bounded, polynomial-time computable functions $n(\cdot) \geq 1$. We stress that the adversary gets d_1 as input, and also gets a $(\cdot)^{d_2} \bmod N$ oracle.

5 Results, discussion and proof ideas

As indicated above, the question we want to answer, for each of our four two-party signature scheme, is: what assumptions on the underlying primitives suffice to guarantee security of the scheme? In this section we provide the theorem statements, some intuition with regard to why the four different two-party signature schemes are relying for their security on different assumptions about the underlying primitives, and also the main ideas behind our proofs of security. The proofs of the theorems are in Section 6. Definitions of the RSA related computational problems used as assumptions are in Section 4. The results are summarized in Figure 1.

OVERVIEW. The middle column of Figure 1 lists assumptions about RSA and the hash function H, and the strength of the assumptions in this column decreases as one moves down. The security statements regarding our four schemes are in the four corner boxes. The six theorems indicated in Figure 1 can be divided into two sets. Theorems labeling arrows originating in the middle column and ending in one of the four corner boxes represent direct proofs of security for the corresponding two-party signature scheme, and are discussed in Section 5.2. The theorems labeling the leftmost and rightmost arrows of the picture are of a different nature, relating different kinds of two-party schemes to each other, and are discussed in Section 5.1. We begin however with a useful lemma.

Technical lemma. The following lemma implies that a randomly chosen point has a non-negligible probability of being a proper client-share of the decryption exponent, and will be used to justify the correctness of all our simulations. A proof can be found in Section 6.1.

Lemma 5.1 *Suppose $k \geq 1$ is an integer, and N is an RSA modulus satisfying $N < 2^k$. If an integer w is drawn at random from $\{1, \ldots, N\}$ then $\Pr[\, w \in \mathsf{Z}^*_{\varphi(N)} \,]$ is strictly greater than $8/[435 \ln(k)]$.* ▮

5.1 Relations between two-party schemes

As Figure 1 indicates, we show the following: If one of the two-party signature scheme from the client-message family of Figure 2 is secure, then so is the corresponding two-party signature scheme from the common-message family of Figure 2. (That is, if HCS is secure so is MCS, and if HSC is secure so is MSC.) The formal theorem statements are as follows, with proofs in Section 6.2 and Section 6.3, respectively.

Theorem 5.2 *Let* DS *be a base signature scheme, let* SKDS-H *be the corresponding HCS two-party signature scheme, and let* SKDS-M *be the corresponding MCS two-party signature scheme. If* SKDS-H *is secure against one-more forgery, then* SKDS-M *is secure against forgery under chosen-message attack.*

Theorem 5.3 *Let* DS *be a base signature scheme, let* SKDS-H *be the corresponding HSC two-party signature scheme, and let* SKDS-M *be the corresponding MSC two-party signature scheme. If* SKDS-H *is secure against one-more forgery, then* SKDS-M *is secure against forgery under chosen-message attack.*

We note that these results make no assumptions about the underlying schemes other than that the meet the stated security notions. In particular the results are true both in a random oracle model and in a standard model. The feature of these results that we feel is interesting is that they relate schemes of different types (common-message versus client-message) and satisfying different types of security notions (security against forgery under cma versus security against one-more forgery).

These two general results help clarify the high-level picture before we move to analyses of individual schemes. One implication, for example, is that the assumption under which we might hope to prove the security of a client-message scheme from Figure 2 will be at least as strong as the assumptions used to prove the security of the corresponding common-message scheme from Figure 2.

5.2 Security results for the two-party schemes

The minimal assumption. Suppose that the base signature scheme DS, with which all our two-party signature schemes are compatible, is *insecure*. We claim this implies that all four two-party signing schemes are insecure. (Given FG who breaks DS, it is a simple exercise to design, for each of the four two-party signature schemes, a client adversary BC that, using FG as a subroutine, breaks the two-party signature scheme in question, with the definition of "breaking" being consistent with the definitions of Section 3 in either case. We omit the details.) The implication is that the security of the base scheme is the *minimal* assumption under which we can hope to prove security of the two-party signature schemes, and thus, the first question to ask is whether this assumption is also sufficient.

SECURITY OF MCS. The minimal assumption that the base signature scheme DS is secure (against forgery under cma) suffices to prove that the common-message two-party signature scheme MCS is also secure (against forgery under cma). This is as good as it gets, and says that the MCS scheme has a strong security guarantee. The formal theorem statement is the following.

Theorem 5.4 *Let* DS *be a base signature scheme, and* SDS *the corresponding MCS two-party signature scheme. If* DS *is secure against forgery under chosen-message attack, then so is* SDS.

The proof is provided in Section 6.4. To better understand what follows however, it is worth explaining the main ideas of the proof here.

Given BC who breaks the MCS scheme, we want to construct FG who breaks DS. The FG, as usual, runs BC as a subroutine to obtain a forgery M, x that it can output. The main issues are: how can it provide BC with the client share d_c of d, and how can it "simulate" SERVER_{N,e,d_s}?

Since the forger knows neither d nor $\varphi(N)$, it is unlikely that it can come up with d_c always distributed exactly as in the output of $\overline{\text{KG}}$. The strategy it uses is to pick d_c at random from $\{1, \ldots, N\}$, and proceed. Lemma 5.1 shows that d_c is correctly distributed with (low but) non-negligible probability, which turns out to be enough.

The forger cannot be expected to come up with d_s satisfying $d_c d_c \equiv d \pmod{\varphi(N)}$, since otherwise it could factor N, so it cannot simulate SERVER_{N,e,d_s} directly. Its strategy is to use its access to the sign-oracle $\text{SIGN}_{N,d}(\cdot)$. When BC initiates an interaction with SERVER_{N,e,d_s} on message M and first flow x_c, the forger responds with $x = \text{SIGN}_{N,d}(M)$. (We are omitting some details, like the fact that the forger needs to check whether or not x_c is correct, since if not it must return $\perp$. See Section 6.4.)

SECURITY OF MSC. The assumption that the base signature scheme DS is secure does not seem enough, however, to guarantee the security of any of the other three schemes. In particular, merely having the server go first, as in MSC, is enough to provide BC with information that might go beyond that provided by the signature itself. Specifically, in MSC, BC as usual knows d_c, but now can also obtain $H(M)^{d_s}$ for a value M of its choice, and mere knowledge of the final signature $H(M)^{d_c d_s} \bmod N$ does *not* seem to provide knowledge of $H(M)^{d_s} \bmod N$ to a party knowing d_c. In the technicalities of the proofs, this translates to saying that the simulation of SERVER_{N,e,d_s} based on access of FG to the sign-oracle, as above, no longer seems possible.

This does not mean that we know of a weakness in the MSC scheme. It does mean, however, that its security relies on stronger assumptions about the underlying primitives than those that sufficed to prove security of MCS. Our result is that the security of MSC can be proved assuming that the RSA-STI is hard and H is a random oracle. The formal statement is the following. The proof is provided in Section 6.5.

Theorem 5.5 *Let* KG *be an RSA key generation algorithm, let* DS *be the corresponding base signature scheme in the random oracle model, and let* SDS *be the corresponding MSC two-party signature scheme in the random oracle model. If the RSA family is one-way with respect to* KG, *then* SDS *is secure against forgery under chosen-message attack.*

The invocation of the random oracle model of course puts the assumptions here into a different class (cf. [6]). However, one should note that all known proofs of security for the "hash-then-decrypt" style base signature scheme that we are considering, under the assumption that RSA-STI is hard, assume that H is a RO [3, 8]. (There are alternative base schemes close in style [16, 9], but their adaptation to a two-party setting has not been investigated.) Thus in terms of pragmatic provable-security, the assumption that RSA-STI is hard and H is a RO may be considered a relatively natural strengthening of the assumption that the base signature scheme is secure.

Why does this change in assumptions suffice to guarantee security? As we saw above, the concern with MSC is that the value $H(M)^{d_s} \bmod N$ that Bc can obtain from the server provides information over and above $H(M)^d \bmod N$. If H is a random oracle, however, it can be shown that $H(M)^{d_s} \bmod N$ is not in fact extra information, by exploiting the idea of the proof of security of the FDH (Full Domain Hash) signature scheme from [3], as follows.

Given Bc who breaks the MSC scheme, we want to construct an inverter In who, given N, e, y, outputs $y^d \bmod N$. It will pick d_c at random, in the same manner as the forger in the proof of security of MCS discussed above, and run Bc on inputs N, e, d_c, itself providing answers to the hash oracle queries, and simulating the responses of $\textsc{Server}_{N,e,d_s}$. The inverter can respond to hash-oracle query M with a point of the form $r^{ed_c} \bmod N$, having itself picked r at random. This means that $H(M)^{d_s} \equiv r \pmod N$, and so the inverter can provide $H(M)^{d_s}$ to Bc if needed. The inverter seeks to obtain $y^d \bmod N$ by ensuring that $H(M) = y$ for the message M that Bc outputs in the forgery. (The difficulty of this last goal not being compatible with the simulation strategy is resolved via a guess as to the hash-oracle corresponding to the forgery [3]. The strategy of [8] can be used to improve the concrete security of the reduction, but for simplicity in this paper we are not discussing concrete security issues.) For details see Section 6.5.

Security of the client-message schemes. Denying the server knowledge of the message being signed results in Bc being able to obtain y^{d_s} for a value y which it can choose at will (in HSC) or almost at will (in HCS), meaning it effectively has (for HSC) or almost has (for HCS) an oracle for the function $(\cdot)^{d_s} \bmod N$. (In HCS, there is some restriction on the use of the oracle, because Bc must be able to provide $y^{d_c} \bmod N$ when invoking the oracle at y.) In contrast, in the common-message schemes, Bc could only obtain values of this oracle on points y of the form $H(M)$, having first had to commit to M; this tied the use of the oracle more closely to the signing function of the base signature scheme and lead to our being able to show (under somewhat different conditions for MCS and MSC) that, effectively, the oracle did not add much power over and above signing power.

We observe that the HCS and HSC schemes are analogous to Chaum's RSA-based blind-signature scheme [7], both with regard to the security goal (security against one-more-forgery) and the protocol issues discussed above. (In Chaum's scheme the forger effectively has access to an oracle for the function $(\cdot)^d \bmod N$, and wins if it succeeds in one-more forgery.) It is natural to look in that direction for analysis ideas. No proof of security for Chaum's scheme under the standard assumption that RSA-STI is hard has appeared, even in a random oracle model, and furthermore there seems little reason to expect one, since the security relies on properties of RSA that seem to go beyond those reflected in the RSA-STI problem. The situation for HCS and HSC is unlikely to be better. An analysis approach that seems productive in such settings, and was pursued for Chaum's scheme in [1], is to formulate, or distill out, the RSA-related computational problems that seem to underly the scheme, and prove security assuming these problems are hard. In this manner, at least one better understands what one is assuming. We use the problems formulated in [1] in our current setting.

Security of HCS. We prove the security of HCS under the same assumption that was used in [1] to prove the security of Chaum's blind signature scheme, namely that the RSA-CTI (RSA Chosen Target Inversion) problem is hard and H is a random oracle. The formal theorem statement is the following. The proof is provided in Section 6.5.

Theorem 5.6 *Let* KG *be an RSA key generation algorithm, let* DS *be the corresponding base signature scheme in the random oracle model, and let* SDS *be the corresponding HCS two-party signature scheme in the random oracle model. If the RSA-CTI problem is hard with respect to* KG, *then* SDS *is secure against one-more-forgery.*

Chaum's blind signature scheme is not a two-party scheme. By saying that the assumptions underlying it suffice also to prove security of HCS, we are saying that the splitting of the key has not created security weaknesses beyond those already present in a scheme in which the adversary effectively gets an RSA-inversion oracle. The proof of Theorem 5.6 indicates how this is possible, and we now discuss some of its ideas.

Given Bc attacking HCS, we are trying to build an adversary A for the RSA-CTI problem. As in all our proofs, A picks d_c at random, and Lemma 5.1 is used to argue that this choice is a properly distributed client-share with non-negligible probability. It then runs Bc on inputs N, e, d_c. The main issue is how to handle the simulation of $\textsc{Server}_{N,e,d_s}$, which, as we observed before, is effectively providing a $(\cdot)^{d_s}$ oracle to Bc. The important point is that the flow from the client includes not just y but also x_c, where x_c should equal $y^{d_c} \bmod N$. Assuming it does, the server's response is $x_s \equiv x_c^{d_s} \equiv y^d \pmod N$, meaning that $(\cdot)^{d_s}$ oracle can be simulated given access to a $(\cdot)^d \bmod N$ oracle, and thus can be simulated by A since it has access to a $(\cdot)^d \bmod N$ oracle. The correctness of this simulation relies on the fact that $\textsc{Server}_{N,e,d_s}$ checks that the client computes x_c correctly (by checking that $x_s^e \equiv y \pmod N$) and that A can perform this check too since it knows d_c. The assumption that H is a random oracle is used in transforming the outputs of Bc into ones properly related to the inputs of A. See Theorem 5.6 and its proof for details.

Security of HSC. Having the server go first introduces an extra security risk by giving the client-adversary more power. In HCS there is some restriction on the use of the $(\cdot)^{d_s} \bmod N$ oracle effectively provided to the client via its access to $\textsc{Server}_{N,e,d_s}$, namely that when calling this oracle on input y, the client must supply $x_c = y^{d_c} \bmod N$. As we saw above, this means that the $(\cdot)^{d_s} \bmod N$ oracle can be simulated via a $(\cdot)^d \bmod N$ oracle. In HSC, there is no such brake put on the client's access to the $(\cdot)^{d_s} \bmod N$ oracle, and it does not seem possible to simulate it given access to a $(\cdot)^d \bmod N$ oracle. Thus, it appears that the HSC scheme relies for its security on assumptions even stronger than the hardness of the RSA-CTI problem. Indeed, this is the first of the four schemes for which we have not found a way to avoid explicitly considering the key-splitting in the assumptions. (For all the other schemes, the assumptions related to the $(\cdot)^d \bmod N$ function, meaning to the original RSA problem.)

We prove the security of HSC based on the assumption that the Split-Key-RSA-CTI problem is hard and H is a random oracle.

Theorem 5.7 *Let* KG *be an RSA key generation algorithm, let* DS *be the corresponding base signature scheme in the random oracle model, and let* SDS *be the corresponding HSC two-party signature scheme in the random oracle model. If the RSA-S-CTI problem is hard with respect to* KG, *then* SDS *is secure against one-more-forgery.*

The proof, which can be found in Section 6.7, is relatively straightforward.

6 Proofs of results in Section 5

6.1 Proof of Lemma 5.1

We will make use of the following lower bound on $\varphi(m)/m$, which can be derived from Rosser and Schoenfeld [25, Theorem 15]:

Lemma 6.1 *If $M \geq 3$ is an integer then*

$$\frac{\varphi(M)}{M} \;\geq\; \frac{1}{29 \ln\ln(M)} \;. \blacksquare$$

Proof of Lemma 5.1: The probability in question is

$$\Pr[\, w \in \mathsf{Z}^*_{\varphi(N)} \,] \;=\; \frac{\varphi(\varphi(N))}{N} \;=\; \frac{\varphi(\varphi(N))}{\varphi(N)} \cdot \frac{\varphi(N)}{N} \;.$$

Since N is an RSA modulus, it must be that $N \geq 15$, implying that $\varphi(N) \geq 3$, and thus we can apply Lemma 6.1, with $M = \varphi(N)$, to lower bound the first term above. For the second term, we use the fact that $\varphi(N) = (p-1)(q-1)$ where p, q are the distinct, odd primes such that $N = pq$. From the above we get

$$\begin{aligned}
\Pr[\, w \in \mathsf{Z}^*_{\varphi(N)} \,] \;&\geq\; \frac{1}{29 \ln\ln(\varphi(N))} \cdot \left(1 - \frac{1}{p}\right) \cdot \left(1 - \frac{1}{q}\right) \\
&\geq\; \frac{1}{29 \ln\ln(\varphi(N))} \cdot \left(1 - \frac{1}{3}\right) \cdot \left(1 - \frac{1}{5}\right) \\
&=\; \frac{8}{435 \ln\ln(\varphi(N))} \\
&\geq\; \frac{8}{435 \ln\ln(N)} \\
&>\; \frac{8}{435 \ln(k)} \;.
\end{aligned}$$

In the last step we used the bound $\ln\ln(N) \leq \ln(k \ln(2)) < \ln(k)$, which is true because we have assumed $N < 2^k$. $\blacksquare$

6.2 Proof of Theorem 5.2

We associate to any polynomial-time client-adversary BcM another polynomial-time client adversary BcH such that for all k we have

$$\mathbf{Adv}^{\text{uf-cma}}_{\text{SKDS-M,BcM}}(k) \;\leq\; \mathbf{Adv}^{\text{omf}}_{\text{SKDS-H,BcH}}(k) \;. \tag{3}$$

The theorem follows.

Client-adversary BcH is initialized with N, e, d_c and has access to Server_{N,e,d_s}. It begins with the following initializations:

```
Set counter $i \leftarrow 0$
Initialize BcM with inputs $N, e, d_c$
```

It now runs BcM, who will initiate some number of interactions with its server. Since MCS is a common-message scheme, BcM must begin an interaction with the server by specifying a message to serve as common input. Client-adversary BcH will play the server role. It will get from BcM a message M and first flow x_c, run the following code, and return to Bc the output:

```
If there is a $j \leq i$ such that $M_j = M$ then return $x_j$
Else
    $y \leftarrow H(M)$
    If $x_c \not\equiv y^{d_c} \pmod{N}$ then return $\perp$
    Else
        $i \leftarrow i+1$ ; $M_i \leftarrow M$ ; $y_i \leftarrow y$ ; $x_{i,c} \leftarrow x_c$
        Send $y_i, x_{i,c}$ to $\textsc{Server}_{N,e,d_s}$ and get back $x_i$
        Return $x_i$
```

Above, BcH is careful to not invoke $\textsc{Server}_{N,e,d_s}$ in cases where the latter would return $\perp$, because fruitless calls to its server count against it in a one-more forgery attack. This is important to the analysis that follows.

Finally,BcM will output a pair (M, x). BcH executes the following:

```
$m \leftarrow i$
$i \leftarrow i+1$ ; $M_i \leftarrow M$ ; $x_i \leftarrow x$.
```

It then outputs the list $(M_1, x_1), \ldots, (M_{m+1}, x_{m+1})$ and halts.

For the analysis, we observe that m is the number of sessions that BcH initiated with $\textsc{Server}_{N,e,d_s}$, because each initiation increments i by one above. Thus, BcH is successful as long as

$$\textsc{Verify}^{H}_{N,e}(M_j, x_j) = 1 \tag{4}$$

for all $j = 1, \ldots, m+1$. Equation (4) is true for $j = 1, \ldots, m$ by virtue of the invocation of $\textsc{Server}_{N,e,d_s}$, and is true for $j = m+1$ if BcM is successful. This completes the proof of Equation (3).

6.3 Proof of Theorem 5.3

This proof is very similar to that of Theorem 5.2. (In fact it is simpler.) We associate to any polynomial-time client-adversary BcM another polynomial-time client adversary BcH such that for all k we have

$$\mathbf{Adv}^{\text{uf-cma}}_{\text{SKDS-M},\text{BcM}}(k) \ \leq\ \mathbf{Adv}^{\text{omf}}_{\text{SKDS-H},\text{BcH}}(k)\ . \tag{5}$$

The theorem follows.

Client-adversary BcH is initialized with N, e, d_c and has access to $\textsc{Server}_{N,e,d_s}$. It begins with the following initializations:

```
Set counter $i \leftarrow 0$
Initialize BcM with inputs $N, e, d_c$
```

It now runs BcM, who will initiate some number of interactions with its server. Since MSC is a common-message scheme, BcM must begin an interaction with the server by specifying a message to serve as common input. Client-adversary BcH will play the server role. It will get from BcM a message M, run the following code, and return to Bc the output:

```
If there is a $j \leq i$ such that $M_j = M$ then return $x_j$
Else
    $i \leftarrow i+1$ ; $M_i \leftarrow M$ ; $y_i \leftarrow H(M)$
    Send $y_i$ to $\textsc{Server}_{N,e,d_s}$ and get back $x_{i,s}$
    Return $x_{i,s}$
```

Finally,BcM will output a pair (M, x). BcH executes the following:

```
$m \leftarrow i$
$i \leftarrow i+1$ ; $M_i \leftarrow M$ ; $x_i \leftarrow x$.
```

It then outputs the list $(M_1, x_1), \ldots, (M_{m+1}, x_{m+1})$ and halts.

For the analysis, we observe that m is the number of sessions that BcH initiated with $\textsc{Server}_{N,e,d_s}$, because each initiation increments i by one above. Thus, BcH is successful as long as

$$\textsc{Verify}^{H}_{N,e}(M_j, x_j) = 1 \tag{6}$$

for all $j = 1, \ldots, m+1$. Equation (6) is true for $j = 1, \ldots, m$ by virtue of the invocation of $\textsc{Server}_{N,e,d_s}$, and is true for $j = m+1$ if BcM is successful. This completes the proof of Equation (5).

6.4 Proof of Theorem 5.4

We associate to any polynomial-time client-adversary Bc a polynomial-time forger Fg such that for all k we have

$$\mathbf{Adv}^{\text{uf-cma}}_{\text{SDS,Bc}}(k) \;\le\; \frac{435\ln(k)}{8} \cdot \mathbf{Adv}^{\text{uf-cma}}_{\text{DS},F}(k)\ . \tag{7}$$

The theorem follows.

Adversary Fg takes as input an RSA public key N, e, and has access to sign-oracle $\textsc{Sign}^{H}_{N,d}(\cdot)$. It begins with the following initializations:

```
Set counter $i \leftarrow 0$
Pick $d_c$ at random from $\{1, \ldots, N\}$
Initialize Bc with inputs $N, e, d_c$
```

Not knowing $\varphi(N)$ it cannot pick d_c at random from $\mathsf{Z}^{*}_{\varphi(N)}$, so instead it picks d_c at random from $\{1, \ldots, N\}$ as above. It now runs Bc, who will initiate some number of interactions with the server. Since MCS is a common-message scheme, Bc must begin an interaction with the server by specifying a message to serve as common input. Adversary Fg will play the server role. It will get from Bc a message M and a flow x_c for the server, un the following code, and return to Bc the output:

```
$i \leftarrow i+1$
$M_i \leftarrow M$ ; $x_{i,c} \leftarrow x_c$
Let $w_{i,c} \leftarrow H(M_i)^{d_c} \bmod N$
If $x_{i,c} \neq w_{i,c}$ then $x_i \leftarrow \perp$ else $x_i \leftarrow \textsc{Sign}^{H}_{N,d}(M_i)$
Return $x_i$
```

Finally, Bc will output a pair M, x, which Fg also outputs and halts.

For the analysis, we consider the following events:

GS	:	$d_c \in \mathsf{Z}^{*}_{\varphi(N)}$
Win	:	$\textsc{Verify}^{H}_{N,e}(M, x) = 1$ and M was not a message with which Bc initialized the server.

We claim that

$$\Pr[\,\mathsf{Win} \mid \mathsf{GS}\,] \;=\; \mathbf{Adv}^{\text{uf-cma}}_{\text{SDS,Bc}}(k)\ . \tag{8}$$

The justification for this is as follows. Suppose event GS is true. Then d_c is uniformly distributed in $\mathsf{Z}^*_{\varphi(N)}$, and is thus distributed identically to the share of d that is issued to $\textsc{Bc}$ by the split-key generation algorithm $\overline{\textsc{kg}}$, so the inputs to $\textsc{Bc}$ are distributed as in the experiment measuring its advantage. Furthermore, the response x_i returned by the forger to flow $x_{i,c}$ is distributed identically to that which would have been returned by the real server. Now we can lower bound the advantage of $\textsc{Fg}$ as follows:

$$\begin{aligned}
\mathbf{Adv}^{\text{uf-cma}}_{\textsc{ds},\textsc{Fg}}(k) &\geq \Pr[\,\mathsf{Win}\,] \\
&\geq \Pr[\,\mathsf{Win} \wedge \mathsf{GS}\,] \\
&= \Pr\left[\,\mathsf{Win} \mid \mathsf{GS}\,\right] \cdot \Pr[\,\mathsf{GS}\,] \\
&= \mathbf{Adv}^{\text{uf-cma}}_{\textsc{sds},\textsc{Bc}}(k) \cdot \Pr[\,\mathsf{GS}\,] \\
&\geq \mathbf{Adv}^{\text{uf-cma}}_{\textsc{sds},\textsc{Bc}}(k) \cdot \frac{8}{435\ln(k)}\ .
\end{aligned}$$

The first inequality is true because if event Win happens, then $\textsc{Fg}$ is certainly successful. The second inequality is obvious, and the following equality is by Equation (8). The last inequality used Lemma 5.1. This completes the proof of Equation (7).

6.5 Proof of Theorem 5.5

We adapt the proof of security of the FDH signature scheme from [3], following the proof of security of the S-RSA protocol from [20]. We associate to any polynomial-time client-adversary $\textsc{Bc}$ a polynomial-time inverter $\textsc{In}$ such that for all k we have

$$\mathbf{Adv}^{\text{uf-cma}}_{\textsc{sds},\textsc{Bc}}(k) \ \leq\ \frac{435\ln(k)}{8} \cdot q(k) \cdot \mathbf{Adv}^{\text{rsa-sti}}_{\textsc{kg},\textsc{In}}(k)\ , \tag{9}$$

where $q(\cdot)$ denotes a polynomially-bounded function such that the number of server-interactions initiated by $\textsc{Bc}$, plus the number of hash-oracle queries it makes, is strictly upper bounded by $q(k)$. The theorem follows.

Adversary $\textsc{In}$ takes as input an RSA public key N, e, security parameter k, and a point $y \in \mathsf{Z}^*_N$. It is seeking to output $y^d \bmod N$. It begins with the following initializations:

```
Set counter $i \leftarrow 0$
Pick $d_c$ at random from $\{1, \ldots, N\}$
Pick $l$ at random from $\{1, \ldots, q(k)\}$
Initialize Bc with inputs $N, e, d_c$
```

It now runs $\textsc{Bc}$, who will initiate some number of interactions with the server, and also make queries to its hash-oracle. A hash-oracle query M is answered by running the following code and returning to $\textsc{Bc}$ the output:

```
HSim($M$)
If there is a $j \leq i$ such that $M_j = M$ then return $H_j$
Else
    $i \leftarrow i + 1$ ; $M_i \leftarrow M$
    $x_{i,s} \overset{R}{\leftarrow} \mathsf{Z}^*_N$ ; $r_i \leftarrow [x^e_{i,s}]^{d_c} \bmod N$
    If $i = l$ then $H_i \leftarrow y$ else $H_i \leftarrow r_i$
    Return $H_i$
```

Since MSC is a common-message scheme, $\textsc{Bc}$ must begin an interaction with the server by specifying a message to serve as common input. Adversary $\textsc{In}$ will play the server role. It will get from $\textsc{Bc}$ a message M, run the following code, and return to $\textsc{Bc}$ the output:

```
SSim($M$)
$H \leftarrow$ HSim($M$)
Let $j$ be the index such that $M = M_j$
If $j = l$ then abort else return $x_{j,s}$
```

The "abort" instruction above means that $\textsc{In}$ halts its entire computation with no output. If this does not happen then eventually $\textsc{Bc}$ outputs a pair M, x. $\textsc{In}$ returns x and halts.

For the analysis, we consider the following events:

GS	:	$d_c \in \mathsf{Z}^*_{\varphi(N)}$
Win	:	$\textsc{Verify}^H_{N,e}(M, x) = 1$ and M was not a message with which $\textsc{Bc}$ initialized the server
NA	:	The abort instruction was never executed by SSim
CG	:	$M_l = M$

We claim that

$$\Pr[\,\mathsf{Win} \mid \mathsf{GS} \wedge \mathsf{NA} \wedge \mathsf{CG}\,] = \mathbf{Adv}^{\text{uf-cma}}_{\text{SDS},\text{Bc}}(k)\ . \tag{10}$$

This and Lemma 5.1 are used in the following estimates:

$$\begin{aligned}
\mathbf{Adv}^{\text{rsa-sti}}_{\text{KG},\,\text{In}}(k) &\geq \Pr[\,\mathsf{Win}\,] \\
&\geq \Pr[\,\mathsf{Win} \wedge \mathsf{GS} \wedge \mathsf{NA} \wedge \mathsf{CG}\,] \\
&= \Pr[\,\mathsf{Win} \mid \mathsf{GS} \wedge \mathsf{NA} \wedge \mathsf{CG}\,] \cdot \Pr[\,\mathsf{GS} \wedge \mathsf{NA} \wedge \mathsf{CG}\,] \\
&= \mathbf{Adv}^{\text{uf-cma}}_{\text{SDS},\text{Bc}}(k) \cdot \Pr[\,\mathsf{GS}\,] \cdot \Pr[\,\mathsf{NA} \wedge \mathsf{CG}\,] \\
&\geq \mathbf{Adv}^{\text{uf-cma}}_{\text{SDS},\text{Bc}}(k) \cdot \frac{8}{435\ln(k)} \cdot \frac{1}{q(k)}\ .
\end{aligned}$$

This completes the proof of Equation (9).

6.6 Proof of Theorem 5.6

We adapt the proof of security of Chaum's RSA-based blind signature scheme from [1]. We associate to any polynomial-time client-adversary $\textsc{Bc}$ a polynomial-time inverter $\textsc{In}$ such that for all k we have

$$\mathbf{Adv}^{\text{omf}}_{\text{SDS},\text{Bc}}(k) \leq \frac{435\ln(k)}{8} \cdot \mathbf{Adv}^{\text{rsa-cti}}_{\text{KG},\,\text{In},\,n}(k)\ , \tag{11}$$

where $n(\cdot) \geq 1$ is a polynomially-bounded, polynomial time computable function such that the number of server-interactions initiated by $\textsc{Bc}$, plus the number of hash-oracle queries it makes, is strictly upper bounded by $n(k)$. The theorem follows.

Adversary $\textsc{In}$ takes as input an RSA public key N, e, security parameter k, and a sequence $y_1, \ldots, y_{n(k)}$ of points in Z^*_N. It has access to a $(\cdot)^d \bmod N$ oracle. It begins with the following initializations:

```
Set counter $i \leftarrow 0$
Pick $d_c$ at random from $\{1, \ldots, N\}$
Initialize Bc with inputs $N, e, d_c$
```

It now runs Bc, who will initiate some number of interactions with the server, and also make queries to its hash-oracle. A hash-oracle query M is answered by running the following code and returning to Bc the output.

```
HSim($M$)
If there is a $j \leq i$ such that $M_j = M$ then return $y_j$
Else
    $i \leftarrow i + 1$ ; $M_i \leftarrow M$
    Return $y_i$
```

Bc will begin each interaction with the server by providing values to serve the role of the client's first protocol flow. (Since this is a client-message scheme, no message is provided by Bc.) Adversary In will play the server role. It will get from Bc the first flow y, x_c, run the following code, and return to Bc the output:

```
SSim($y, x_c$)
If $y^{d_c} \not\equiv x_c \pmod{N}$ then $x \leftarrow \perp$ else $x \leftarrow y^d \bmod N$
Return $x$
```

Finally, Bc will output a sequence $(\overline{M}_1, \overline{x}_1), \ldots, (\overline{M}_{m+1}, \overline{x}_{m+1})$ of pairs. In now executes the following:

```
For $j = 1, \ldots, m + 1$ do
    Run HSim($\overline{M}_j$)
    Let $i$ be such that $\overline{M}_j = M_i$
    $\pi(j) \leftarrow i$
EndFor
```

It then outputs $(m, \pi, \overline{x}_1, \ldots, \overline{x}_{m+1})$ and halts.

For the analysis, we consider the following events:

$$\begin{array}{rcl} \mathsf{GS} & : & d_c \in \mathsf{Z}^*_{\varphi(N)} \\ \mathsf{Win} & : & \textsc{Verify}^H_{N,e}(\overline{M}_j, \overline{x}_j) = 1 \text{ for all } j = 1, \ldots, m, \text{ and} \\ & & \text{Bc initiated at most } m \text{ server-interactions} \end{array}$$

We claim that

$$\Pr[\,\mathsf{Win} \mid \mathsf{GS}\,] = \mathbf{Adv}^{\mathrm{omf}}_{\mathrm{SDS,Bc}}(k) \; . \tag{12}$$

This and Lemma 5.1 are used in the following estimates:

$$\begin{array}{rcl} \mathbf{Adv}^{\mathrm{rsa\text{-}cti}}_{\mathrm{KG,\,In},\,n}(k) & \geq & \Pr[\,\mathsf{Win}\,] \\ & \geq & \Pr[\,\mathsf{Win} \wedge \mathsf{GS}\,] \\ & = & \Pr[\,\mathsf{Win} \mid \mathsf{GS}\,] \cdot \Pr[\,\mathsf{GS}\,] \\ & = & \mathbf{Adv}^{\mathrm{omf}}_{\mathrm{SDS,Bc}}(k) \cdot \Pr[\,\mathsf{GS}\,] \\ & \geq & \mathbf{Adv}^{\mathrm{omf}}_{\mathrm{SDS,Bc}}(k) \cdot \dfrac{8}{435 \ln(k)} \; . \end{array}$$

This completes the proof of Equation (11).

6.7 Proof of Theorem 5.7

This proof is very similar to that of Theorem 5.6. We associate to any polynomial-time client-adversary BC a polynomial-time inverter IN such that for all k we have

$$\mathbf{Adv}^{\mathrm{omf}}_{\mathrm{SDS,BC}}(k) \;\leq\; \mathbf{Adv}^{\mathrm{sk\text{-}rsa\text{-}cti}}_{\mathrm{KG,\,IN},\,n}(k)\;, \tag{13}$$

where $n(\cdot) \geq 1$ is a polynomially-bounded, polynomial time computable function such that the number of server-interactions initiated by BC, plus the number of hash-oracle queries it makes, is strictly upper bounded by $n(k)$. The theorem follows.

Adversary IN takes as input an RSA public key N, e, a value $d_1 \in \mathsf{Z}^*_{\varphi(N)}$, security parameter k, and a sequence $y_1, \ldots, y_{n(k)}$ of points in Z^*_N. It has access to a $(\cdot)^{d_2} \bmod N$ oracle. It begins with the following initializations:

```
Set counter $i \leftarrow 0$
$d_c \leftarrow d_1$
Initialize BC with inputs $N, e, d_c$
```

It now runs BC, who will initiate some number of interactions with the server, and also make queries to its hash-oracle. A hash-oracle query M is answered by running the following code and returning to BC the output.

```
HSim($M$)
If there is a $j \leq i$ such that $M_j = M$ then return $y_j$
Else
    $i \leftarrow i + 1$ ; $M_i \leftarrow M$
    Return $y_i$
```

BC will begin each interaction with the server by providing values to serve the role of the client's first protocol flow. (Since this is a client-message scheme, no message is provided by BC.) Adversary IN will play the server role. It will get from BC the first flow y, run the following code, and return to BC the output:

```
SSim($y$)
$x_s \leftarrow y^{d_2} \bmod N$
Return $x_s$
```

Finally, BC will output a sequence $(\overline{M}_1, \overline{x}_1), \ldots, (\overline{M}_{m+1}, \overline{x}_{m+1})$ of pairs. IN now executes the following:

```
For $j = 1, \ldots, m+1$ do
    Run HSim($\overline{M}_j$)
    Let $i$ be such that $\overline{M}_j = M_i$
    $\pi(j) \leftarrow i$
EndFor
```

It then outputs $(m, \pi, \overline{x}_1, \ldots, \overline{x}_{m+1})$ and halts.

For the analysis, we consider the following event:

Win : $\mathrm{VERIFY}^H_{N,e}(\overline{M}_j, \overline{x}_j) = 1$ for all $j = 1, \ldots, m$, and
BC initiated at most m server-interactions

It is easy to see that

$$\mathbf{Adv}^{\text{rsa-cti}}_{\text{KG, IN}, n}(k) \;\geq\; \Pr[\,\mathsf{Win}\,] \;=\; \mathbf{Adv}^{\text{omf}}_{\text{SDS,BC}}(k)\ ,$$

and this completes the proof of Equation (13).

References

[1] M. Bellare, C. Namprempre, D. Pointcheval and M. Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. Record 2001/002, IACR Cryptology ePrint archive, `http://eprint.iacr.org`. Preliminary version, entitled The power of RSA inversion oracles and the security of Chaum's RSA-based blind signature scheme, in *Financial Cryptography '01*, Lecture Notes in Computer Science Vol. 2339, P. Syverson ed., Springer-Verlag, 2001.

[2] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. *Proceedings of the 1st Annual Conference on Computer and Communications Security*, ACM, 1993.

[3] M. Bellare and P. Rogaway. The exact security of digital signatures—how to sign with RSA and Rabin. *Advances in Cryptology – EUROCRYPT '96*, Lecture Notes in Computer Science Vol. 1070, U. Maurer ed., Springer-Verlag, 1996.

[4] D. Boneh and M. Franklin. Efficient generation of shared RSA keys. *Advances in Cryptology – CRYPTO '97*, Lecture Notes in Computer Science Vol. 1294, B. Kaliski ed., Springer-Verlag, 1997.

[5] C. Boyd. Digital multisignatures. In *Cryptography and Coding*, H. Beker and F. Piper eds., Oxford University Press, 1989, pp. 241–246.

[6] R. Canetti, O. Goldreich and S. Halevi. The random oracle methodology, revisited. *Proceedings of the 30th Annual Symposium on the Theory of Computing*, ACM, 1998. Available as Record 1998/011, IACR Cryptology ePrint archive, `http://eprint.iacr.org`.

[7] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology – CRYPTO '82*, Lecture Notes in Computer Science, D. Chaum, R. Rivest, and A. Sherman, editors, Plenum Press, 1983.

[8] J. Coron. On the exact security of full domain hash. *Advances in Cryptology – CRYPTO '00*, Lecture Notes in Computer Science Vol. 1880, M. Bellare ed., Springer-Verlag, 2000.

[9] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. *ACM Transactions on Information and System Security*, 3(3):161–185, Aug. 2000.

[10] A. De Santis, Y. Desmedt, Y. Frankel and M. Yung. How to share a function securely. *Proceedings of the 26th Annual Symposium on the Theory of Computing*, ACM, 1994.

[11] Y. Desmedt and Y. Frankel. Threshold cryptosystems. *Advances in Cryptology – CRYPTO '89*, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989.

[12] Y. Frankel, P. MacKenzie and M. Yung. Robust efficient distributed RSA key generation. *Proceedings of the 30th Annual Symposium on the Theory of Computing*, ACM, 1998.

[13] R. Ganesan. Yaksha: Augmenting Kerberos with public-key cryptography. *Proceedings of the ISOC Network and Distributed Systems Security Symposium*, 1995.

[14] R. Ganesan. Yaksha: Towards reusable security infrastructures. *Ph.D thesis*, Johns Hopkins University, 1996.

[15] R. Ganesan and Y. Yacobi. A secure joint signature and key-exchange system. *Bellcore TM-24531*, October 1994.

[16] R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. *Advances in Cryptology – EUROCRYPT '99*, Lecture Notes in Computer Science Vol. 1592, J. Stern ed., Springer-Verlag, 1999.

[17] R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin. Robust and efficient sharing of RSA functions. *Advances in Cryptology – CRYPTO '96*, Lecture Notes in Computer Science Vol. 1109, N. Koblitz ed., Springer-Verlag, 1996.

[18] N. Gilboa. Two-party RSA key generation. *Advances in Cryptology – CRYPTO '99*, Lecture Notes in Computer Science Vol. 1666, M. Wiener ed., Springer-Verlag, 1999.

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

[20] P. MacKenzie and M. Reiter. Networked cryptographic devices resilient to capture. *Proceedings of the IEEE Symposium on Security and Privacy*, IEEE, 2001.

[21] P. MacKenzie and M. Reiter. Two-party generation of DSA signatures. *Advances in Cryptology – CRYPTO '01*, Lecture Notes in Computer Science Vol. 2139, J. Kilian ed., Springer-Verlag, 2001.

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

[23] G. Poupard and J. Stern. Generation of shared RSA keys by two parties. *Advances in Cryptology – ASIACRYPT '98*, Lecture Notes in Computer Science Vol. 1514, D. Pei ed., Springer-Verlag, 1998.

[24] RSA Laboratories. PKCS#1: RSA Cryptography Standard. `http://www.rsalabs.com/pkcs/pkcs-1/index.html`.

[25] J. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois J. of Math*, 6:64–94, 1962.

[26] SingleSignOn.Net. `http://www.singlesignon.net`.

[27] A. Yao. How to Generate and Exchange Secrets. *Proceedings of the 27th Symposium on Foundations of Computer Science*, IEEE, 1986.

A Relations between RSA related problems

Note that the RSA-STI and RSA-CTI$_1$ problems are identical, so that RSA-CTI can be considered a natural extension of RSA-STI. For further information regarding the RSA-CTI problem and its relation to other problems, refer to [1].

The natural question with regard to the new Split-Key-RSA-CTI problem we have introduced is its relation to the RSA-CTI problem. A simple observation is that splitting the key cannot make the adversary's task harder, meaning that if the Split-Key-RSA-CTI problem is hard then so is the RSA-CTI problem. The following Proposition provides the formal statement and is proved below.

Proposition A.1 Let KG be an RSA key generator and let $n : \mathsf{N} \to \mathsf{N}$ be a (polynomially bounded, polynomial time computable) function of the security parameter such that $n(k) \geq 1$ for all $k \in \mathsf{N}$. If the Split-Key-RSA-CTI$_n$ problem is hard with respect to KG then the RSA-CTI$_n$ problem is hard with respect to KG. ▌

The more interesting question is whether the hardness of the RSA-CTI problem implies the hardness of the Split-Key-RSA-CTI problem. We do not see how to prove this in general. However, we note that one can prove that the hardness of RSA-CTI$_n$ implies the hardness of Split-Key-RSA-CTI$_n$ in the special case that $n(\cdot) = 1$. The reason this case is different is that the sk-rsa-cti adversary is allowed no oracle queries. The reason it is still somewhat non-trivial is that the sk-rsa-cti adversary does, however, receive d_1 as input. The result is summarized in the following Proposition and proved below. Note that RSA-CTI$_1$ is the same as RSA-STI, so we are saying that the three problems RSA-STI, RSA-CTI$_1$ and Split-Key-RSA-CTI$_1$ are all equivalent in hardness.

Proposition A.2 Let KG be an RSA key generator. If the RSA-CTI$_1$ problem is hard with respect to KG, then the Split-Key-RSA-CTI$_1$ problem is hard with respect to KG. ▌

This tells us that the splitting of the key results in (possibly) more power to the adversary due to a combination of reasons: the fact that it has d_1 as input and the fact that it has a $(\cdot)^{d_2} \bmod N$ oracle.

Proof of Proposition A.1: We associate to any polynomial-time rsa-cti adversary A a polynomial-time sk-rsa-cti adversary B such that for all k we have

$$\mathbf{Adv}^{\text{rsa-cti}}_{\text{KG},\,A,\,n}(k) \;\leq\; \mathbf{Adv}^{\text{sk-rsa-cti}}_{\text{KG},\,B,\,n}(k)\;. \tag{14}$$

The Proposition follows. Adversary B has input $N, e, d_1, k, y_1, \ldots, y_{n(k)}$ and access to a $(\cdot)^{d_2} \bmod N$ oracle. It runs A on inputs $N, e, k, y_1, \ldots, y_{n(k)}$. When A makes a query v to its $(\cdot)^{d} \bmod N$ oracle, B queries $v^{d_1} \bmod N$ to its $(\cdot)^{d_2} \bmod N$ oracle to get a response w, and returns w to A as the answer to A's oracle query. B outputs whatever A outputs. It is easy to see that the simulation of A's $(\cdot)^{d} \bmod N$ oracle that B provides is always correct, and hence that Equation (14) is true. We omit the details. ▌

Proof of Proposition A.2: We associate to any polynomial-time sk-rsa-cti adversary B a polynomial-time rsa-cti adversary A such that for all k we have

$$\mathbf{Adv}^{\text{sk-rsa-cti}}_{\text{KG},\,B,\,1}(k) \;\leq\; \frac{435\ln(k)}{8}\cdot \mathbf{Adv}^{\text{rsa-cti}}_{\text{KG},\,A,\,1}(k)\;. \tag{15}$$

The Proposition follows. Adversary A has input N, e, k, y_1. (It also has access to $(\cdot)^{d} \bmod N$ oracle but is allowed zero queries to it, so we can imagine the oracle is absent.) Not knowing $\varphi(N)$ it cannot pick d_1 at random from $\mathsf{Z}^*_{\varphi(N)}$, so instead it picks d_1 at random from $\{1, \ldots, N\}$, and runs B on inputs N, e, d_1, k, y_1. It outputs whatever B outputs. (Since B too is allowed zero oracle queries, the output is obtained directly.) The simulation of B is correct as long as d_1 is uniformly distributed in $\mathsf{Z}^*_{\varphi(N)}$, and this is true with probability at least $8/[435\ln(k)]$ by Lemma 5.1. Equation (15) follows. We omit the details. ▌