On the Security of RSA Encryption in TLS

Jakob Jonsson and Burton S. Kaliski Jr.

RSA Laboratories, 20 Crosby Drive, Bedford, MA 01730, USA
{jjonsson, bkaliski}@rsasecurity.com

Abstract. We show that the security of the TLS handshake protocol based on RSA can be related to the hardness of inverting RSA given a certain "partial-RSA" decision oracle. The reduction takes place in a security model with reasonable assumptions on the underlying TLS pseudo-random function, thereby addressing concerns about its construction in terms of two hash functions. The result is extended to a wide class of constructions that we denote *tagged key-encapsulation mechanisms*.

Keywords: key encapsulation, RSA encryption, TLS.

1 Introduction

One of the most popular methods for establishing secret information between two parties with no prior shared secret is the handshake protocol used in the Secure Sockets Layer (SSL) [15] and Transport Layer Security (TLS) [10] protocols (which we will refer to jointly as TLS). These protocols support a variety of algorithms (called "cipher suites"). In the suite of interest to this paper, the handshake protocol is based on the RSA-PKCS-1v1_5 (abbreviated RSA-P1) encryption scheme introduced in the PKCS #1 v1.5 [31] specification, which in turn is based on the RSA trapdoor permutation [30].

Due to their widespread use, both the TLS handshake protocol and the underlying encryption scheme RSA-P1 have been subject to a significant amount of cryptanalysis. A number of weaknesses in RSA-P1 for general message encryption have been found, including the results given in [3, 5, 6, 8]. These results suggest that RSA-P1 must be equipped with certain countermeasures to provide an adequate level of security.

Briefly, the common case of the TLS protocol we will analyze has the following form. A *server* has a public key / private key pair. A *client* establishes a secret s with the server through the following key agreement scheme (omitting certain details):

1. The client and server select a new *nonce* ρ, to which they both contribute.
2. The client generates a *pre-master secret* r, encrypts r with the server's public key under RSA-P1 to obtain a ciphertext y, and sends y to the server. The server decrypts y to recover r.
3. The client and the server both derive a *master secret* t from r.
4. The client and the server compute separate tags z and z' from t and ρ, exchange the tags, and verify them.

5. If the tags are correct, the client and the server both derive a shared secret s from the master secret t.

Only the server has a public key, and hence only the server is authenticated in this scheme. The scheme follows reasonable design principles, such as including a nonce for freshness and a tag for assurance that the client knows the pre-master secret [24]. However, we are not aware of any formal security proof relating the difficulty of "breaking" this scheme to any underlying problem, e.g., RSA.

To facilitate a proof, we will model the interaction between the client and the server as a *tagged key-encapsulation mechanism* (TKEM), which may be viewed as an extension to key encapsulation mechanisms as defined in [28]. The client's steps are considered as an *encryption operation* that produces a ciphertext, a tag, and a secret from the nonce. A *decryption operation* corresponding to the server's steps produces the same secret from the ciphertext, the tag, and the nonce if the tag is correct. (We omit the tag computed by the server in this model as it is not needed for the proof, and in any case is no more helpful to an adversary than the one computed by the client.) The security of the key agreement scheme is thus transformed to the indistinguishability of the TKEM against a chosen-ciphertext attack.

Using this model, we show that the security of the TKEM underlying TLS can be related via a reasonably tight reduction to the hardness of inverting RSA with the assistance of a "partial-RSA decision oracle". This oracle takes as input an RSA ciphertext y and a bit string r of length 384 bits and checks whether r is equal to the last 384 bits of the RSA decryption of y. While based on a stronger assumption than the corresponding proofs for RSA-OAEP [2, 16], RSA-OAEP+ [27], RSA-KEM [32, 1, 28], RSA-REACT [23], and RSA-GEM [7], this is the first security proof relating an RSA-P1-based application to the RSA problem.

We consider two TKEMs in this paper. The first TKEM is based on a single pseudo-random function, which we model as a random oracle. However, TLS actually uses the xor of two pseudo-random functions, each based on a different hash function, to address the risk that one of the pseudo-random functions might turn out to be weak. The second TKEM we consider follows this design, and we model only one of the pseudo-random functions as a random oracle and assume no specific properties of the other. As a result, we are able to show that the TLS handshake is secure even if one of the hash functions turns out to be weak. This addresses a concern [18] about the TLS pseudo-random function construction.

2 Basic Concepts

In Section 2.1, we provide basic concepts and definitions related to trapdoor mappings; the special case RSA-P1, intended for use within the TKEM2 mechanism introduced in Section 3.2, is defined in Section 2.2. In Section 2.3 we define plaintext-checking oracles instrumental for the security reductions.

Notation

A *bit string* is an ordered sequence of elements from $B = \{0,1\}$. For $n \geq 0$, $B^n = \{0,1\}^n$ denotes the set of bit strings of length n. An *octet* is an element in B^8. Let B^* denote the set of all bit strings. Bit strings and octet strings are identified with the integers they represent in base 2 and 2^8, respectively. $\mathbb{Z}_n$ denotes the set $\{0,\ldots,n-1\}$; $\mathbb{Z}_n$ is the additive group of integers modulo n. To denote that an element a is chosen uniformly at random from a finite set A, we write $a \stackrel{\mathsf{R}}{\leftarrow} A$.

2.1 Randomized Trapdoor Mappings

Here we give a brief introduction to randomized trapdoor mappings, generalizing the concept of trapdoor permutations; a trapdoor mapping is invertible but not necessarily deterministic. Let k be a security parameter. For each k, let $\mathcal{E}_k$ be a finite family of pairs (E,D) with the property that E is a randomized and reversible algorithm with inverse D. E takes as input an element r in a set $R = R_E$ and returns an element y in a set $Y = Y_E$, possibly via randomness generated within the algorithm; we will write $y \leftarrow E(r)$. D is a deterministic algorithm $Y \rightarrow R \cup \{\phi\}$ such that $D(y) = r$ if $y \leftarrow E(r)$ for some $r \in R$ and $D(y) = \phi$ otherwise. Each output y from E corresponds to at most one input r. y is *valid* if $D(y) \neq \phi$ and *invalid* otherwise. We assume that the running time of each of E and D is polynomial in k.

Let $\mathcal{G}$ be a probabilistic polynomial-time (PPT) algorithm that on input 1^k (i.e., k uniformly random bits) outputs a pair $(E,D) \in \mathcal{E}_k$. $\mathcal{G}$ is a *trapdoor mapping generator*. An *E-inverter* $\mathcal{I}$ is an algorithm that on input (E,y) tries to compute $D(y)$ for a random $y \in Y$. $\mathcal{I}$ has success probability $\epsilon = \epsilon(k)$ and running time $T = T(k)$ if

$$\Pr\left((E,D) \leftarrow \mathcal{G}(1^k), r \stackrel{\mathsf{R}}{\leftarrow} R_E, y \leftarrow E(r) : \mathcal{I}(E,y) = r\right) \geq \epsilon$$

and the running time for $\mathcal{I}$ is at most T. In words, $\mathcal{I}$ should be able to compute $D(y)$ with probability ϵ within time T, where (E,D) is derived via the trapdoor mapping generator and y is random. $\mathcal{I}$ solves the *E problem*.

$\mathcal{E}_k$ is a *trapdoor mapping family* with respect to (ϵ,T) if the E-problem is (ϵ,T)-*hard*, meaning that there is no E-inverter with success probability ϵ within running time T. The individual mapping E is referred to as a *trapdoor mapping.* A *trapdoor permutation* is a deterministic trapdoor mapping E with $Y_E = R_E$.

An RSA permutation $f : \mathbb{Z}_N \rightarrow \mathbb{Z}_N$ is defined in terms of an *RSA public key* (N,e) as $f(x) = x^e \bmod N$. N is the *RSA modulus*, a product of two secret integer primes p and q, while e is an odd (typically small) integer such that $\gcd(e,(p-1)(q-1)) = 1$. RSA permutations are widely believed to be trapdoor permutations. This means that it is presumably hard to compute $f^{-1}(y)$ on a random input $y \in \mathbb{Z}_N$ provided N is large enough and generated at random. Yet, given secret information (e.g., the prime factors of N), the inverse $f^{-1}(y)$ is easy to compute.

2.2 RSA-PKCS-1v1_5

Here we describe the specific trapdoor mapping RSA-P1 (RSA-PKCS-1v1_5) introduced in PKCS #1 v1.5, which is based on the RSA permutation. For the purposes of this paper, the input to the RSA-P1 encryption operation has a fixed length; in PKCS #1 v1.5, the input may have a variable length.

Let l_N be the octet length of the RSA modulus N. The encryption operation takes as input an element $r \in B^{8l_r}$, where $l_r \leq l_N - 11$; put $k_r = 8l_r$. (In TLS, $l_r = 48$ and $k_r = 384$.) The trapdoor mapping RSA-P1 is defined as follows; 00 and 02 are octets given in hexadecimal notation. Note that P is an octet string of length $l_N - l_r - 3 \geq 8$ consisting of nonzero octets.

RSA-P1-Encrypt(r)

– $P \stackrel{\mathsf{R}}{\leftarrow} \left(B^8 \setminus \{00\}\right)^{l_N - l_r - 3}$;
– $x \leftarrow 00\|02\|P\|00\|r$;
– $y \leftarrow x^e \bmod N$;
– Return the integer y.

Aligning with the terminology in Section 2.1, an RSA-P1 inverter solves the *RSA-P1 problem*, whereas an RSA inverter solves the *RSA problem*.

2.3 Plaintext-Checking and Partial-RSA Decision Oracles

One property of many trapdoor mappings as defined in Section 2.1 is that it is presumably hard to tell for a given pair (r, y) whether $D(y) = r$ if D is secret. For example, RSA-P1 defined in Section 2.2 appears to have this property. The reason is that there might be many possibilities for $E(r)$ for each r as soon as E is randomized. While this may appear to be an attractive feature of a trapdoor mapping, in our setting it is in fact a drawback. Namely, to reduce an E-inverter to a chosen-ciphertext attack against the tagged key-encapsulation mechanisms introduced in Section 3, we must be able to simulate an oracle that on input (r, y) tells whether $D(y) = r$. There is no generic solution to this problem.

To address this concern, we make use of the *plaintext-checking oracle* concept introduced in [22, 23]. This oracle, which we denote PO_E, takes as input a pair $(r, y) \in R \times Y$ and checks whether $r = D(y)$. If this is true, the oracle outputs the bit 1. Otherwise it outputs the bit 0. The plaintext-checking oracle is correct on each input with probability 1.

For the specific RSA trapdoor permutation f, we introduce also a *partial-RSA decision oracle*. For any integer $k_0 < k$ (k is the bit length of the modulus), this oracle DO_{f,k_0} takes as input a string $r \in B^{k_0}$ and a ciphertext $y \in \mathbb{Z}_N$ and checks whether

$$f^{-1}(y) \bmod 2^{k_0} = r \ .$$

Thus this oracle compares a string r with a "partial" RSA inverse of y.

An interesting question is whether this oracle helps an adversary invert RSA. If k_0 is almost as large as k, the oracle clearly does not help since the adversary can simulate it efficiently, either by guessing or by applying the reduction technique of Coppersmith [5]. The latter case can accommodate k_0 down to the range of $k(1-1/e)$, where e is the RSA encryption exponent. If k_0 is small (say, smaller than 80), the oracle clearly does help: After a brute-force search using the oracle at most 2^{k_0} times, the adversary will be able to determine the last k_0 bits of the RSA inverse. Applying the method described in Appendix A, the adversary will then easily determine the whole of the inverse via another $k-k_0$ applications of the oracle.

For TLS, we have $k_0 = 192$ or 384 (depending on how we model the underlying pseudo-random function) and typically $k \geq 1024$, which implies that neither of the above extreme cases apply. Our conjecture is that k_0 is large enough to render the oracle useless.

An algorithm is $PO(q)$-*assisted* (resp. $DO(q)$-assisted) if it has access to a plaintext-checking oracle PO (resp. decision oracle DO) that accepts at most q queries. A PO_E-assisted E-inverter solves the *gap-E problem*. For example, a $PO_{\mathrm{RSA-P1}}$-assisted RSA-P1 inverter solves the *gap-RSA-P1 problem*. A DO_{f,k_0}-assisted RSA inverter solves the *gap-partial-RSA problem* with parameter k_0.

3 Tagged Key-Encapsulation Mechanisms

In this section we discuss the concept of tagged key-encapsulation mechanisms (TKEM), which are useful for modeling key agreement schemes such as the one in the TLS handshake protocol. Section 3.1 is devoted to a TKEM defined in terms of a trapdoor mapping and a pseudo-random function approximating the mechanism underlying TLS, while Section 3.2 concentrates on identifying the specific RSA-P1-based TKEM within TLS. (The Diffie-Hellman [11] version of the TLS handshake, which is based on ElGamal [13] key agreement as defined in [21], can also be modeled with a TKEM; however, we do not address that version here.)

A TKEM consists of an *encryption operation* and a *decryption operation*. The encryption operation TKEM-Encrypt takes as input a public key K_{pub}, a nonce ρ, and possibly some other parameters and returns a triple (y, z, s), where y is the *ciphertext*, z is the *tag*, and s is the *secret*. The decryption operation TKEM-Decrypt takes as input a private (secret) key K_{priv}, a ciphertext y, a tag z, a nonce ρ, and possibly some other parameters and returns the secret s or "Error" if the ciphertext is not valid or the tag is inconsistent with the nonce and the other information. Each new application of TKEM-Decrypt requires a nonce that has not been used before.

As noted above, the security of the key agreement scheme in the TLS handshake can be transformed to the indistinguishability of the underlying TKEM. In particular, the key agreement scheme is secure against an adversary sending ciphertexts to the server if the underlying TKEM is secure against an adversary sending ciphertexts to a decryption oracle simulating TKEM-Decrypt. It is clear that the latter adversary is at least as strong as the former adversary as she is allowed to select the entire nonce herself (as long as it is new).

3.1 TKEM1

We introduce the tagged key-encapsulation scheme TKEM1 as a first approximation to the TLS handshake. TKEM1 is defined in terms of a trapdoor mapping E; write $R = R_E$ and $Y = Y_E$. Let $h : B^* \times B^* \times \mathbb{Z} \to B^*$ be a pseudo-random function; $h(r, \sigma, l)$ is a string of length l (there might be restrictions on the sizes of the inputs and outputs). Fix parameters k_s, k_t, k_z, k_δ, and k_ρ; these parameters denote bit lengths of different strings used within TKEM1. Let $\Delta : B^* \times B^* \times B^* \to b^{k_\delta}$ be an arbitrary function; we will refer to the output from Δ as a *digest* (this reflects the way Δ is used within TLS). Δ is part of the tag derivation construction. Let Q_s, Q_t, and Q_z be three distinct strings such that no string is a prefix of any of the others. This is to ensure that the inputs to the different applications of h are distinct.

The TKEM1 encryption operation takes as input a nonce $\rho \in B^{k_\rho}$ and a label $L \in B^*$. ρ provides *freshness*; each application of TKEM1 uses a new nonce. L contains public information that is intended to be integrity-protected by TKEM1. The TKEM1 encryption operation is defined as follows.

TKEM1-Encrypt(ρ, L)

– $r \overset{\mathsf{R}}{\leftarrow} R$;
– $y \leftarrow E(r)$;
– $t \leftarrow h(r, Q_t \| \rho, k_t)$;
– $z \leftarrow h(t, Q_z \| \Delta(\rho, L, y), k_z)$;
– $s \leftarrow h(t, Q_s \| \rho, k_s)$;
– Return the ciphertext y, the tag z, and the secret s.

The corresponding decryption operation TKEM1-Decrypt is defined in the obvious way: First, $r = D(y)$ is recovered (if $D(y) = \phi$, then an error message is returned). Second, a tag z' is computed from r, ρ, L, and y. If $z' = z$, then the secret s is recovered and returned; otherwise, an error message is returned.

Since the output length of h is uniquely determined by the prefix of the second input (the prefix is either Q_s, Q_t, or Q_z) we will suppress the third argument of h below.

In practice, to thwart implementation attacks such as [3] and [20], the decryption operation should not output "Error" immediately if $D(y) = \phi$. Instead, the operation should proceed with a new r generated uniformly at random. With high probability, z will not match z', and "Error" will be output at the end, without revealing whether $D(y) = \phi$ or z' is incorrect.

3.2 TKEM2

TKEM2 is the specific TKEM used within TLS and may be viewed as a special case of TKEM1. However, while our security model for TKEM1 will assume that h is a strong pseudo-random function, the corresponding function in TLS, denoted Ω, has a seemingly more fragile structure, which will require special treatment.

Ω is defined as follows on input $(w, \sigma, l) \in B^{2*} \times B^* \times \mathbb{Z}$ (B^{2*} is the set of strings with an even bit length). Write $w = w^{\mathrm{L}} \| w^{\mathrm{R}}$, where the bit length of each of the strings w^{L} and w^{R} is half the bit length of w. Then

$$\Omega(w, \sigma, l) = g(w^{\mathrm{L}}, \sigma, l) \oplus h(w^{\mathrm{R}}, \sigma, l) ,$$

where g and h are pseudo-random functions defined in terms of HMAC [19] based on MD5 [26] and SHA-1 [29], respectively. We refer the interested reader to [10] for details. The rationale for the Ω construction is to achieve a reasonable amount of security even if one of the underlying pseudo-random functions turns out to be weak.

The function Δ outputs the concatenation of a SHA-1 hash value and an MD5 hash value of a string derived from the input nonce, label, and ciphertext; L corresponds to all messages in the handshake other than ρ and y. However, our results do not depend on the function Δ.

Let RSA-P1 denote the encryption scheme specified in Section 2.2. Let N be an RSA modulus and let e be an RSA public exponent; define $f(x) = x^e \bmod N$. With notation as in TKEM1, we fix $R = B^{384}$, $k_t = k_r = 384$ and $k_z = 96$, whereas k_s depends on the chosen cipher suite. An application of TKEM2 requires a version number v, a bit string of length 16. For simplicity, we assume that the version number is fixed. The nonce ρ is the concatenation of two strings ρ_1 and ρ_2 (one provided by the client and the other by the server). Each string is 32 octets long; the first 4 octets denote the number of seconds since 1/1/1970, and the rest are generated at random. Given the nonce ρ and a label L, the TKEM2 encryption operation proceeds as follows; the decryption operation is defined in alignment with TKEM1-Decrypt.

TKEM2-Encrypt(ρ, L)

- $r_0 \leftarrow v \in \{0,1\}^{16}$; $r_1 \xleftarrow{\mathsf{R}} \{0,1\}^{k_r - 16}$; $r = r_0 \| r_1$;
- $y \leftarrow$ RSA-P1(r);
- $t \leftarrow \Omega(r,$ "master secret"$\| \rho, k_t)$;
- $z \leftarrow \Omega(t,$ "client finished"$\| \Delta(\rho, L, y), k_z)$;
- $s \leftarrow \Omega(t,$ "key expansion"$\| \rho, k_s)$;
- Return the ciphertext y, the tag z, and the secret s.

4 Reduction from the Gap-Partial-RSA Problem to the Gap-RSA-P1 Problem

Before proceeding with security proofs for TKEM1 and TKEM2, we show how a PO_E-assisted RSA-P1-inverter ($E =$ RSA-P1 with parameter k_r) can be extended to a DO_{f,k_r}-assisted RSA inverter. Thus we reduce the gap-partial-RSA problem with parameter k_r to the gap-RSA-P1 problem.[1]

[1] The related problem of finding a reduction from a PO_E-assisted RSA inverter to a PO_E-assisted RSA-P1-inverter is substantially more complex but can be solved via lattice reduction; consider the generalization in [9] of the approach in [16].

Due to the shape of RSA-P1, the two oracles PO_E and DO_{f,k_r} act equivalently except on inputs (r, y) such that $D(y) = \phi$. In this case PO_E always outputs the bit 0, while DO_{f,k_r} outputs the bit 1 if $f^{-1}(y) \bmod 2^{k_r} = r$.

Let Y_0 be the set of valid ciphertexts y. Such ciphertexts have the property that $f^{-1}(y) = 00\|02\|P\|00\|r$ for some $r \in B^{k_r}$ and some string P of nonzero octets. Assuming that k_r and the bit length k of the RSA modulus are multiples of 8, the size of $Y_0/2^k$ is

$$2^{-24}\left(1-2^{-8}\right)^{(k-k_r-24)/8} > 2^{-24}\left(2^{-1/177}\right)^{(k-k_r-24)/8} = 2^{-24-(k-k_r-24)/1416} ,$$

which implies that $|Y_0|/|Y|$ is at least $2^{-24-(k-k_r-24)/1416}$, where $Y = \mathbb{Z}_N$. This is a lower bound on the probability that a uniformly random $y \in \mathbb{Z}_N$ is a valid RSA-P1 ciphertext. The reduction is as follows; the result is easily generalized to other RSA-based trapdoor mappings where the padding is prepended to the plaintext r. The proof is given in Appendix A.

Theorem 1. *Let E be an RSA-P1 trapdoor mapping, let k be the bit length of the RSA modulus N, and let q be a parameter. Assume that there is a $PO_E(q)$-assisted E-inverter with running time at most T' and success probability at least ϵ'. Then there is a $DO_{f,k_r}((q+1)(k-k_r+1))$-assisted f-inverter with running time at most T that is successful with probability at least ϵ, where*

$$\epsilon = \epsilon' \cdot \frac{|Y_0|}{|Y|};$$
$$T = T' + O((q+1)(k-k_r+1) \cdot T_{DO_{f,k_r}}) + O((q+1)k^3) ;$$

$T_{DO_{f,k_r}}$ is the running time for the partial-RSA decision oracle DO_{f,k_r}.

5 Security of TKEM1

5.1 TKEM1 Security Model

We define an attack model employing an adversary against TKEM1 who is given free access to a *decryption oracle*; hence we consider the family of adaptive chosen-ciphertext attacks (CCA2; see [17]). The task for the adversary is to distinguish a secret s_0^* corresponding to a certain ciphertext y^* with parameters (z^*, ρ^*, L^*) from a random string.

The decryption oracle responds to a query (y, z, ρ, L) with the corresponding secret $s = h(t, Q_s\|\rho)$ if y is a valid ciphertext and the tag z is correct ($t = h(D(y), Q_t\|\rho)$); otherwise, the oracle responds with a generic error message. The decryption oracle accepts any query, except that previous nonces must not be reused.

We will not make any specific assumptions about the digest function Δ. However, the function h will be instantiated as a random oracle. Thus the adversary has no information about $h(r, \sigma)$ unless she sends the *query* (r, σ) to an oracle instantiating h. The h-oracle responds to queries with strings chosen uniformly at random and independent from earlier queries and responses, except that a string that is repeatedly queried to the oracle should have the same response every time. To distinguish between different h-oracle queries, we let h_s-queries be queries prefixed by Q_s, while h_t-queries and h_z-queries are queries prefixed by Q_t and Q_z, respectively.

The attack experiment runs as follows. First, the adversary is given a trapdoor mapping E generated via a trapdoor mapping generator $\mathcal{G}$. The adversary is allowed to send queries to the h-oracle and the decryption oracle during the entire attack. At any time of the attack – but only once – the adversary sends a query (ρ^*, L^*) to a *challenge generator*[2]. Here, ρ^* must not be part of any previous or later decryption query. The challenge generator applies the TKEM1-ENCRYPT operation, producing a ciphertext y^*, a tag z^*, and a secret s_0^*. In addition, the generator selects a uniformly random string s_1^* and flips a fair coin b. The generator returns (y^*, z^*) and s_b^*; thus the response depends on b.

At the end, the adversary outputs a bit b'. The *distinguishing advantage* of the adversary is defined as $\Pr(b' = b) - \Pr(b' \neq b) = 2\Pr(b' = b) - 1$; the probability is computed over all possible trapdoor mappings for a given security parameter k. The adversary is an *IND-CCA2* adversary [17, 25] (IND = indistinguishability).

5.2 Reduction from a PO_E-Assisted E-Inverter to an IND-CCA2 Adversary Against TKEM1

In the full version of this paper, we show how to reduce a PO_E-assisted E-inverter to an IND-CCA2 adversary against TKEM1. If E is equal to RSA-P1, the result is easily translated into a reduction from the gap-partial-RSA problem to the security of TKEM1 via Theorem 1. Let $k_r = \lceil \log_2 |R| \rceil$. The result is as follows; the proof is suppressed in this extended abstract.

Theorem 2. *Let q_s, q_t, q_z, q_D be parameters. Assume that there is an IND-CCA2 adversary against TKEM1 with advantage at least ϵ' within running time T' making at most q_s, q_t, q_z, and q_D number of h_s-queries, h_t-queries, h_z-queries, and decryption queries, respectively. Then there is a $PO_E(2q_t)$-assisted E-inverter with running time at most T that is successful with probability at least ϵ, where*

$$\epsilon = \epsilon' - ((q_D + 2)(q_D + q_z) + q_s) \cdot 2^{-k_t} - q_D \cdot 2^{-k_z} \; ;$$
$$T = T' + O(2q_t \cdot T_{PO_E}) + \tau \; ;$$

T_{PO_E} is the running time for the plaintext-checking oracle PO_E, while τ is the time needed for $O(q_D + q_t + q_s + q_z)$ elementary table operations.

Remark. If E is deterministic, the E-inverter can implement PO_E directly by applying E; thus we may replace T_{PO_E} with the running time for an application of E. Note that when E is the RSA trapdoor permutation, we obtain a tight

[2] Some authors denote this generator as an *encryption oracle*; since we find this notation somewhat confusing, we have chosen another term.

reduction from the ordinary RSA problem to the security of TKEM1. This result aligns with prior work on RSA-based key encapsulation mechanisms in [28, 23].

6 Security of TKEM2

In this section we restrict our attention to the TKEM2 mechanism defined in Section 3.2. Let f be an RSA permutation; $f(x) = x^e \bmod N$, where (N, e) is an RSA public key. Let E denote the corresponding RSA-P1 trapdoor mapping with $R_E = B^{k_r}$. We want to relate the security of TKEM2 to the gap-partial-RSA problem with parameter k_r.

Ideally, we would like to analyze the security of TKEM2 assuming that Ω is a random oracle. Indeed, with this assumption TKEM2 would be a special case of TKEM1. Unfortunately however, as the discussion in [18] indicates, this assumption does not seem appropriate for the specific PRF in TLS. First, the xor construction weakens the pseudo-randomness of the PRF output in terms of the input. Second, MD5 is known to have certain theoretical weaknesses [4, 12], which makes the random oracle assumption even less reasonable.

Instead, we will assume that only h based on SHA-1 is a random oracle; g based on MD5 will be treated as an ordinary function with no specific properties. It is possible to obtain a proof in case g is a random oracle and h is an ordinary function, but the reduction and the underlying decision oracle will be slightly different. Introduce six functions $g_t, h_t, g_s, h_s, g_z, h_z$ defined as

$$h_t(r^{\mathrm{R}}, \rho) = h(r^{\mathrm{R}}, \text{“master secret”} \| \rho, k_t) \ ;$$
$$h_s(t^{\mathrm{R}}, \rho) = h(t^{\mathrm{R}}, \text{“key expansion”} \| \rho, k_s) \ ;$$
$$h_z(t^{\mathrm{R}}, \delta) = h(t^{\mathrm{R}}, \text{“client finished”} \| \delta, k_z) \ ,$$

and analogously for g_t, g_s, and g_z. Let $h_t^{\mathrm{L}}(r^{\mathrm{R}}, \rho)$ denote the first $k_r/2$ bits of $h_t(r^{\mathrm{R}}, \rho)$ and let $h_t^{\mathrm{R}}(r^{\mathrm{R}}, \rho)$ denote the last $k_r/2$ bits. Use the corresponding notation for the two halves of $g_t(r^{\mathrm{L}}, \rho)$. With notations as in Section 3.2, note that

$$s = g_s\left(g_t^{\mathrm{L}}(r^{\mathrm{L}}, \rho) \oplus h_t^{\mathrm{L}}(r^{\mathrm{R}}, \rho), \rho\right) \oplus h_s\left(g_t^{\mathrm{R}}(r^{\mathrm{L}}, \rho) \oplus h_t^{\mathrm{R}}(r^{\mathrm{R}}, \rho), \rho\right) \ ;$$

the tag z satisfies a similar relation in terms of g_t, h_t, g_z and h_z.

Now, assume that h_t, h_s, and h_z are random oracles, while the other functions are just ordinary functions. Since we apply a random oracle only to the second half of the RSA-P1 input r, we will relate the security of TKEM2 to a gap-partial-RSA inverter with parameter $k_r/2$ rather than k_r. This is due to the PRF construction; with a stronger PRF, we would be able to give a security proof in terms of DO_{f,k_r} or PO_E. The result is as follows; a proof sketch is given in Appendix B (see the full version of this paper for complete details).

Theorem 3. *Let q_t, q_s, q_z, q_D be parameters. Assume that there is an IND-CCA2 adversary against TKEM2 with advantage at least ϵ' within running time*

T' making at most q_t, q_s, q_z, and q_D queries to the h_t-oracle, h_s-oracle, h_z-oracle, and decryption oracle, respectively. Let $q' = 2q_t + (k - k_r/2)(q_D + 1)$. Then there is a $DO_{f,k_r/2}(q')$-assisted RSA inverter with running time at most T that is successful with probability at least ϵ, where

$$\epsilon = c \cdot \left(\epsilon' - ((q_D + 2)(q_D + q_z) + q_s) \cdot 2^{-k_t/2} - q_D \cdot 2^{-k_z}\right) \ ;$$
$$T = T' + O\left(q' \cdot T_{DO_{f,k_r/2}}\right) + O((q_D + 1)k^3) + \tau \ ;$$

$c = 2^{-40-(k-k_r-24)/1416}$. $T_{DO_{f,k_r/2}}$ is the running time for the partial-RSA decision oracle $DO_{f,k_r/2}$, while τ is the time needed for $O(q_D + q_t + q_s + q_z)$ elementary table operations.

Remark. The RSA problem is *randomly self-reducible* [14]: If an RSA inverter fails on an input y, one may run the inverter on new random inputs of the form $y' = (f(\alpha) \cdot y) \bmod N$ until the inverter is successful; $f^{-1}(y)$ is easy to derive from $f^{-1}(y') = (\alpha \cdot f^{-1}(y)) \bmod N$. The constant c in Theorem 3 is a lower bound on the probability that the inverter will not trivially fail just because the target ciphertext y is not a valid RSA-P1 ciphertext. Via $1/c$ applications of the random self-reducibility principle, the inverter in Theorem 3 can hence be translated into a new inverter with success probability and running time approximately $\left(1 - (1 - c)^{1/c}\right)\epsilon' > \epsilon'/2$ and T'/c, respectively.

7 Discussion and Conclusion

We have provided a security reduction from a variant of the RSA problem to the tagged key-encapsulation scheme based on RSA-P1 used within TLS. As a byproduct we have addressed the concern about the underlying function Ω. In particular, our proof holds even if MD5 is insecure.

An important aspect of any security reduction is what it implies in practice. Here, we might start with the typical assumption that the RSA problem, for 1024-bit keys, requires about 2^{80} steps to break. Assuming that the gap-partial-RSA problem is just as hard, and with typical parameters, Theorem 3 indicates that no IND-CCA2 adversary against TKEM2 can succeed in fewer than about 2^{40} steps. Of course, this does not mean that there is an attack that succeeds in so few steps, and perhaps there is a better proof than ours that results in a better bound.[3]

The security of RSA-P1 used within TLS depends on the difficulty of the gap-partial-RSA problem. We conjecture that for typical parameters the gap-partial-RSA problem is as hard as the RSA problem. However, this problem needs further study. Indeed, an efficient solution to the problem might well lead to an effective chosen-ciphertext attack on TLS servers. The study of this problem is thus important in practice as well as in theory.

[3] In fact, we can get a better bound simply by changing assumptions: if we assume that the gap-partial-RSA problem *for a random, valid RSA-P1 ciphertext* is as hard as the RSA problem, then the bound will again be about 2^{80}.

Though the security reduction for TKEM2 does not say very much for typical key sizes, the reduction does show, at least intuitively, that there is some strength in the way the RSA algorithm is employed in TLS. It also helps show how the algorithm might be employed better. First, we need to get a tighter bound. This can be done by reducing the number of fixed octets in the input to the RSA operation (there are currently up to five fixed octets). Second, we need to get to the ordinary RSA problem. This can be done by processing all of the input to the RSA operation with a secure function h. Essentially, TLS should use TKEM1 rather than TKEM2.

Security protocols have sometimes been designed with proofs of security in mind, and sometimes only according to "reasonable design principles." TLS was designed originally according to the latter philosophy, but we have shown that the former benefit is achieved as well, though this is somewhat accidental. In general we would argue for an approach that considers both philosophies at the same time.

Acknowledgements

We thank Håkan Andersson, Ari Juels, Mike Szydlo and the anonymous referees for valuable comments on preliminary versions of this paper. Phil Rogaway and Johan Håstad provided helpful feedback on aspects of this research. Don Johnson's observation about the hash function construction in TLS motivated our focus on TKEM2; Johnson also provided useful suggestions for the final version of this paper.

References

1. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. *Proceedings of the First Annual Conference on Computer and Communications Security*. ACM, 1993.
2. M. Bellare and P. Rogaway. Optimal Asymmetric Encryption – How to Encrypt with RSA. *Advances in Cryptology – Eurocrypt '94*, pp. 92 – 111. Springer Verlag, 1994.
3. D. Bleichenbacher. Chosen Ciphertext Attacks against Protocols Based on the RSA Encryption Standard PKCS #1. *Advances in Cryptology – Crypto '98*, pp. 1 – 12. Springer Verlag, 1998.
4. B. den Boer and A. Bosselaers. Collisions for the Compression Function of MD5. *Advances in Cryptology – Eurocrypt '93*, pp. 293-304. Springer Verlag, 1994.
5. D. Coppersmith. Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. *Journal of Cryptology*, 10, pp. 233 – 260, 1997.
6. D. Coppersmith, M. Franklin, J. Patarin and M. Reiter. Low-Exponent RSA with Related Messages. *Advances in Cryptology – Eurocrypt '96*, pp. 1 – 9. Springer Verlag, 1996.
7. J.-S. Coron, H. Handschuh, M. Joye, P. Paillier, D. Pointcheval and C. Tymen. GEM: a Generic Chosen-Ciphertext Secure Encryption Method. *Topics in Cryptology – CT-RSA 2002*, pp. 263 – 276. Springer Verlag, 2002.

8. J.-S. Coron, M. Joye, D. Naccache and P. Paillier. New Attacks on PKCS #1 v1.5 Encryption. *Advances in Cryptology – Eurocrypt 2000*, pp. 369 – 379. Springer Verlag, 2000.
9. J.-S. Coron, M. Joye, D. Naccache and P. Paillier. Universal Padding Schemes for RSA. *Advances in Cryptology – Crypto 2002*, these proceedings.
10. T. Dierks and C. Allen. *IETF RFC 2246: The TLS Protocol Version 1.0*. January 1999.
11. W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT–22(6), pp. 644 – 654. November 1976.
12. H. Dobbertin. *Cryptanalysis of MD5 Compress*. Presented at the rump session of Eurocrypt '96, May 14, 1996.
13. T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, IT–31(4), pp. 469 – 472. July 1985.
14. J. Feigenbaum. Locally Random Reductions in Interactive Complexity Theory. *Advances in Computational Complexity, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 13, pp. 73–98, 1993.
15. A. O. Freier, P. Karlton, and P. C. Kocher. *The SSL Protocol Version 3.0*. Netscape Communications Corp., November 1996.
16. E. Fujisaki, T. Okamoto, D. Pointcheval and J. Stern. RSA-OAEP Is Secure under the RSA Assumption. *Advances in Cryptology – Crypto 2001*, pp. 260 – 274. Springer Verlag, 2001.
17. S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28 (2). April 1984.
18. D. B. Johnson. *Theoretical Security Concerns with TLS Use of MD5*. Contribution to ANSI X9F1 working group. June 21, 2001.
19. H. Krawczyk, M. Bellare and R. Canetti. *IETF RFC 2104: HMAC: Keyed-Hashing for Message Authentication*. February 1997.
20. J. Manger. A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1 v2.0. *Advances in Cryptology – Crypto 2001*, pp. 260 – 274. Springer Verlag, 2001.
21. A. J. Menezes, P. C. van Oorschot and S. A. Vanstone. *Handbook of Applied Cryptography*, CRC Press, 1996.
22. T. Okamoto and D. Pointcheval. The Gap-Problems: a New Class of Problems for the Security of Cryptographic Schemes. *Proceedings of the 2001 International Workshop on Practice and Theory in Public Key Cryptography (PKC'2001)*, pp. 104 – 118. Springer-Verlag, 2001.
23. T. Okamoto and D. Pointcheval. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. *Topics in Cryptology – CT-RSA 2001*, pp. 159 – 175. Springer Verlag, 2001.
24. L. C. Paulson. Inductive analysis of the Internet protocol TLS. *ACM Transactions on Information and System Security*, 2(3), pp. 332 – 351. August 1999.
25. C. Rackoff and D. R. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. *Advances in Cryptology – Crypto '91*, pp. 433 – 444. Springer-Verlag, 1992.
26. R. Rivest. *IETF RFC 1321: The MD5 Message-Digest Algorithm*. April 1992.
27. V. Shoup. OAEP Reconsidered. *Advances in Cryptology – Crypto 2001*, pp. 239 – 259. Springer Verlag, 2001.
28. V. Shoup. *A Proposal for an ISO Standard for Public Key Encryption*. Preprint, December 2001. Available from eprint.iacr.org/2001/112.

29. National Institute of Standards and Technology (NIST). *Draft FIPS 180-2: Secure Hash Standard.* Draft, May 2001.
30. R. Rivest, A. Shamir and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2), pp. 120 - 126. February 1978.
31. RSA Laboratories. *PKCS #1 v1.5: RSA Encryption Standard*. November 1993.
32. Y. Zheng and J. Seberry. Practical Approaches to Attaining Security Against Adaptively Chosen Ciphertext Attacks. *Advances in Cryptology – Crypto '92*, pp. 292-304. Springer Verlag, 1992.

A Proof of Theorem 1

First we show that the partial-RSA decision oracle DO_{f,k_r} can simulate the E plaintext-checking oracle PO_E. As a consequence, DO_{f,k_r} is at least as strong as PO_E.

Lemma 1. *Let E be an RSA-P1 trapdoor mapping with parameter k_r. Given a pair (r, y), PO_E can be perfectly simulated via $k - k_r + 1$ queries to DO_{f,k_r}, where k is the bit length of the RSA modulus. The running time for the simulation is bounded by*

$$O((k - k_r + 1)T_{DO_{f,k_r}}) + O(k^3) \ .$$

Proof of Lemma 1. Suppose that we are given a pair (r, y). Send (r, y) to DO_{f,k_r}. If the response is 0, then either y is not a valid ciphertext (i.e., $D(y) = \phi$) or y is the encryption of a plaintext different from r. In this case, simulate PO_E by responding with 0.

If the response is 1, proceed as follows. We want to determine the whole of $x = f^{-1}(y)$. To achieve this, define $y_0 = y$ and $y_i = 2^e y_{i-1} \bmod N$ for $0 < i \leq k - k_r$. Each y_i can be computed in time $O(k^2)$; $2^e \bmod N$ can be precomputed in time $O(\log_2 e \cdot k^2)$. Put $x_0 = x$ and

$$x_i = f^{-1}(y_i) = 2x_{i-1} \bmod N = 2^i x \bmod N \ ;$$

define $r_i = x_i \bmod 2^{k_r+i}$. Note that

$$r_{k-k_r} = x_{k-k_r} = 2^{k-k_r} x \bmod N \ ,$$

which implies that x can be determined from r_{k-k_r}. We use an induction argument to demonstrate how to determine r_{k-k_r} from r_0 in $k - k_r$ steps, each of complexity $O(k^2) + O(T_{DO_{f,k_r}})$; $O(k^2)$ bounds the time needed to compute y_i from y_{i-1}. Assume that we know r_{i-1}, where i is an integer between 1 and $k - k_r$. Since $x_i = 2x_{i-1} \bmod N$, r_i is either equal to $2r_{i-1}$ or $(2r_{i-1} - N) \bmod 2^{k_r+i}$. To distinguish between the cases, send $(2r_{i-1}, y_i)$ to the decision oracle. If the response is 1, then $r_i = 2r_{i-1}$; otherwise, $r_i = (2r_{i-1} - N) \bmod 2^{k_r+i}$. Thus r_{k-k_r} can be determined via $k - k_r$ applications of the decision oracle.

Once we know r_{k-k_r}, we can easily determine x via $k - k_r$ halving operations modulo N. Now check whether x is valid. Respond with 1 if this is true and 0 otherwise; it is clear that the simulation is perfect, which concludes the proof. □

Proof of Theorem 1. Suppose that we are given a random integer $y^* = f(x^*)$, where x^* is unknown. Generate a random integer $\alpha < N$. With probability $|Y_0|/|Y|$, the integer $y' = \alpha^e \cdot y^* \bmod N$ is a valid encryption of a string r'. Assume that there is an E-inverter that with probability ϵ' outputs the string r'. Thus with probability $\epsilon' \cdot |Y_0|/|Y|$, we have recovered the last k_r bits of $x' = \alpha x^* \bmod N$. (To increase the probability of success, this procedure can be repeated with different values of α until a plaintext is found.) By Lemma 1, each application of PO_E can be replaced with $k - k_r + 1$ applications of DO_{f,k_r}.

Given that we know the last k_r bits r' of x', we want to determine the whole of x', and thereby the whole of x^*. This is done using the method in the proof of Lemma 1 via $k - k_r$ applications of DO_{f,k_r}. The time bound in the theorem includes the time needed to perform $q(k - k_r + 1) + k - k_r < (q+1)(k - k_r + 1)$ DO_{f,k_r}-oracle queries plus a bound on the time needed for the arithmetic computations in the simulation of PO_E and the computations $y' = c^e \cdot y^* \bmod N$ and $x^* = c^{-1}x' \bmod N$ above. □

B Proof Sketch of Theorem 3

Let y^* be the target ciphertext. We deduced in Section 4 that the probability that y^* is a valid RSA-P1 ciphertext is lower-bounded by $2^{-24-(k-k_r-24)/1416}$. However, recall that there are 16 fixed bits in the RSA-P1 input r. This implies that the probability that y^* is a valid TKEM2 ciphertext is lower-bounded by $2^{-40-(k-k_r-24)/1416}$. This is equal to the factor c in the theorem. From now on, assume that y^* is valid.

At the beginning of the attack, the RSA inverter $\mathcal{I}$ generates random strings s_0^*, s_1^*, and z^*. There are additional strings related to the target ciphertext:

$$u^{*\mathrm{L}} \| u^{*\mathrm{R}} = g_t(r^{*\mathrm{L}}, \rho^*) \ ;$$
$$v^{*\mathrm{L}} \| v^{*\mathrm{R}} = h_t(r^{*\mathrm{R}}, \rho^*) \ ;$$

note that

$$s_0^* = g_s(u^{*\mathrm{L}} \oplus v^{*\mathrm{L}}, \rho^*) \oplus h_s(u^{*\mathrm{R}} \oplus v^{*\mathrm{R}}, \rho^*) \ ; \quad (1)$$
$$z^* = g_z(u^{*\mathrm{L}} \oplus v^{*\mathrm{L}}, \delta^*) \oplus h_z(u^{*\mathrm{R}} \oplus v^{*\mathrm{R}}, \delta^*) \ , \quad (2)$$

where $\delta^* = \Delta(\rho^*, L^*, y^*)$. $\mathcal{I}$ does not have to generate $v^{*\mathrm{L}}$ or $v^{*\mathrm{R}}$ in advance, but at the end of the simulation the identities (1) and (2) must be satisfied. Note that ρ^* and L^* are not known yet since they are not provided by the adversary until the challenge generator is queried.

We need to demonstrate how to simulate the oracles corresponding to h_t, h_s, and h_z. $\mathcal{I}$ responds to an h_t-query (r^{R}, ρ) as follows. First, $\mathcal{I}$ checks whether the query is old; in that case the output is already defined. Otherwise, $\mathcal{I}$ sends (r^{R}, y^*) to the partial-RSA decision oracle. If the decision oracle outputs 1, we can get the rest of the inverse $f^{-1}(y^*)$ via $k - k_r/2$ additional queries, following Lemma 1. In this case, $\mathcal{I}$ wins and exits. If the decision oracle outputs 0, $\mathcal{I}$ generates a random string v as the response to the query.

$\mathcal{I}$ responds to an h_s-query (t^{R}, ρ) in the straightforward manner, generating a random output (unless the query is old). $\mathcal{I}$ responds to an h_z-query (t^{R}, δ) in an analogous manner.

$\mathcal{I}$ proceeds as follows on a decryption query (y, z, ρ, L); recall that ρ is different for each decryption query and not equal to ρ^*. First, $\mathcal{I}$ sends (r^{R}, y) to the decision oracle for each string r^{R} such that (r^{R}, ρ) is a previous h_t-query. If the decision oracle outputs 0 on all queries, then $\mathcal{I}$ returns "Error". If the decision oracle outputs 1 for some r^{R}, then $\mathcal{I}$ proceeds as follows.

- Extract the entirety of $x = f^{-1}(y)$ via $k - k_r/2$ queries to the decision oracle using the approach in the proof of Lemma 1.
- If x is not a valid P1 encoding, then output "Error" and exit.
- With $r = x \bmod 2^{k_r}$, compute the corresponding tag z' via the appropriate query to the h_z-oracle combined with the relevant evaluation of g_z.
- If $z = z'$, then compute the corresponding secret s via the appropriate query to the h_s-oracle combined with the relevant evaluation of g_s, output s, and exit. Otherwise, output "Error" and exit.

In the following discussion, some technical details are omitted; see the full version of this paper for a rigorous treatment. It is easily seen that there are only two possible simulation failures:

First, there might be an inconsistency between the simulation of the oracles and the simulation of the challenge generator. This can be the case only if $t^{*\mathrm{R}} = u^{*\mathrm{R}} \oplus v^{*\mathrm{R}}$ is part of an h_s- or h_z-query. Hence, let tBad be the event that there is an h_s- or h_z-query from the adversary or from the decryption oracle including $t^{*\mathrm{R}}$. The probability of tBad is bounded as follows. Note that there are at most $2q_D + q_s + q_z$ different h_s- and h_z-queries. Moreover, note that the simulation is independent of $t^{*\mathrm{R}}$ (i.e., all responses to the adversary are independent of $t^{*\mathrm{R}}$). This implies that the probability of tBad is at most

$$(2q_D + q_s + q_z) \cdot 2^{-k_t/2} \ . \tag{3}$$

Second, some valid ciphertext (y, z, ρ, L) might be erroneously rejected; let $\mathsf{BadReject}$ be the event that this is the case. This event occurs if there is a valid decryption query (y, z, ρ, L) such there is no matching previous h_t-query (r^{R}, ρ). Let (t^{R}, δ) be the input to h_z corresponding to this decryption query. If t^{R} is not part of a previous h_z-query, then the probability that the tag z is valid is 2^{-k_z}. The probability that t^{R} is part of a previous h_z-query is at most $(q_D+q_z)\cdot 2^{-k_t/2}$. This implies that the probability of $\mathsf{BadReject}$ is at most

$$q_D(q_D + q_z) \cdot 2^{-k_t/2} + q_D \cdot 2^{-k_z} \tag{4}$$

(this is a very rough bound but sufficient for our purposes).

It can be shown that the inverter will be successful if the adversary is successful and neither of tBad and $\mathsf{BadReject}$ occurs. Combining (3) and (4), rearranging, and multiplying with c, we obtain the desired probability.