Security Proof of "Efficient and Leakage-Resilient Authenticated Key Transport Protocol Based on RSA"

SeongHan Shin, Kazukuni Kobara, and Hideki Imai

Research Center for Information Security, AIST,
1-18-13, Sotokannda, Chiyoda-ku, Tokyo 101-0021 Japan
seonghan.shin@aist.go.jp, {kobara,imai}@iis.u-tokyo.ac.jp

May 16th, 2006

Abstract. In this paper, we prove the security of the RSA-AKE protocol [9] in the random oracle model. The proof states that the RSA-AKE protocol is secure against an adversary who gets the client's stored secret *or* the server's RSA private key.[1]
To our best knowledge, the RSA-AKE protocol is the most efficient among their kinds (i.e., RSA and password based AKE protocols). The other security properties and efficiency measurements of the RSA-AKE protocol remain the same as in [9].

[1] The protocol is the same as [9], but we corrected the security proof partially. The attacks appeared in [10] are no longer available in the proof since the adversary has access to either the client's stored secret or the server's private key, not both of them.

1 The Model and Security Definitions

In this section we introduce an extended model building on [1, 5] and security definitions for the LR-AKE security and perfect forward secrecy.

We denote by $\mathcal{C}$ and $\mathcal{S}$ two parties that participate in a key exchange protocol P. Each of them may have several instances called oracles involved in distinct, possibly concurrent, executions of P where we denote $\mathcal{C}$ (resp., $\mathcal{S}$) instances by $\mathcal{C}^I$ (resp., $\mathcal{S}^J$), or by U in case of any instance. For the j-th session in our protocol, the party $\mathcal{C}$ remembers a low-entropy secret pw drawn from a small dictionary of password $\mathbb{D}_{\mathsf{Password}}$, whose cardinality is D, and holds another secret α_j on *insecure* devices along with the counterpart's RSA public key (e, N). On the other hand, the party $\mathcal{S}$ stores a verification data p_j and its RSA private key (d, N). Here we suppose a far more powerful adversary (rather than an active one, considered in [3, 6, 1], who has the entire control of the network) by giving additional access to the Leak oracle that simulates *insecure* devices of the party $\mathcal{C}$ and *imperfect* server $\mathcal{S}$. Let us show the capability of adversary $\mathcal{A}$ each query captures:

- $\mathsf{Execute}(\mathcal{C}^I, \mathcal{S}^J)$: This query models passive attacks, where the adversary gets access to honest executions of P between the instances $\mathcal{C}^I$ and $\mathcal{S}^J$ by eavesdropping.
- $\mathsf{Send}(U, m)$: This query models active attacks by having $\mathcal{A}$ send a message to instance U. The adversary $\mathcal{A}$ gets back the response U generates in processing the message m according to the protocol P. A query $\mathsf{Send}(\mathcal{C}^I, \mathtt{Start})$ initializes the key exchange protocol, and thus the adversary receives the initial flow the party $\mathcal{C}$ should send out to the party $\mathcal{S}$.
- $\mathsf{Reveal}(U)$: This query handles the misuse of a session key (e.g., use in a weak symmetric-key encryption) by any instance U. The query is only available to $\mathcal{A}$ if the instance actually holds a session key and the latter is released to $\mathcal{A}$.
- $\mathsf{Leak}(U)$: This query handles the leakage of "stored" secrets by any instance U. The adversary $\mathcal{A}$ gets back the secrets ($\alpha_j, (e, N)$ and (d, N)) where the former (resp., the latter) is released if the instance corresponds to $\mathcal{C}^I$ (resp., $\mathcal{S}^J$). The query is available to $\mathcal{A}$ since the stored secrets might be leaked out due to a bug of the system or physical limitations in the sense that they should be stored on insecure devices all the time.
- $\mathsf{Test}(U)$: This oracle is used to see whether or not the adversary can obtain some information on the challenge session key by giving a hint on the latter. The Test-query can be asked at most once by the adversary $\mathcal{A}$ and is only available to $\mathcal{A}$ if the instance U is "fresh" (see below). This query is answered as follows: one flips a private coin $b \in \{0, 1\}$ and forwards the corresponding session key SK ($\mathsf{Reveal}(U)$ would output) if $b = 1$, or a random value except the session key if $b = 0$.

Definition 1 (Freshness) *We say that an instance is* fresh *(or has a* fresh *session key) if the following conditions hold: (1) the instance has computed and accepted a session key; (2) no* $\mathsf{Corrupt}$*-query has been asked by* $\mathcal{A}$ *before the session key is accepted; and (3) no* Reveal*-query has been asked to the party* $\mathcal{C}$ *nor its partner* $\mathcal{S}$ *in the instance.*

The aim of the adversary is to break the privacy of the session key (a.k.a., semantic security) in the context of executing P. The LR-AKE security is defined by the game $\mathbf{Game}^{\mathsf{lr-ake}}(\mathcal{A}, P)$, in which the adversary $\mathcal{A}$ is provided with random coin tosses, some oracles and then is allowed to invoke any number of queries as described above, in any order. When playing this game, the ultimate goal of the adversary is to guess the bit b in Test-query by outputting this guess b'. We denote LR-AKE advantage, by $\mathsf{Adv}_P^{\mathsf{lr-ake}}(\mathcal{A}) = 2\Pr[b = b'] - 1$, as the probability that $\mathcal{A}$ can correctly guess the value of b. We formally define the LR-AKE security; this will be necessary for stating meaningful results about our protocol.

Definition 2 (LR-AKE Security) *A protocol* P *is said to be LR-AKE secure if, when adversary* $\mathcal{A}$ *(with access to* Leak *oracle) asks* q_s *queries to* Send *oracle and passwords are chosen from a dictionary of size* D*, the adversary's advantage* $\mathsf{Adv}_P^{\mathsf{lr-ake}}(\mathcal{A})$ *in attacking the protocol* P *is bounded by*

$$O(q_s/D) + \epsilon(\cdot), \tag{1}$$

for a negligible function $\epsilon(\cdot)$ *in a security parameter. The first term represents the fact that the adversary can do no better than guess a password during each query to* Send *oracle.*

For the security notion of perfect forward secrecy, one has to account for a new type of query, the $\mathsf{Corrupt}$-query, which models the compromise of the involving parties by adversary $\mathcal{A}$.

– $\mathsf{Corrupt}(U)$: This oracle is used to see whether or not the disclosure of "long-term" secrets of the involving parties does compromise the semantic security of session keys from previous sessions (even though that compromises the authenticity and thus the security of new sessions). With $\mathsf{Corrupt}$-query to instance U, the adversary $\mathcal{A}$ gets back the long-term secrets (pw and its relevant secret p_j) but does not get any internal data.

Definition 3 (Perfect Forward Secrecy) *Suppose an adversary $\mathcal{A}$ with the ability to make the $\mathsf{Corrupt}$-query as well as the other queries described above. A protocol P is said to provide perfect forward secrecy if the adversary's advantage, denoted by $\mathsf{Adv}_P^{\mathsf{pfs-ake}}(\mathcal{A}) = 2\Pr[b = b'] - 1$, in attacking P is negligible in a security parameter.*

1.1 Computational Assumption

Next we define the standard RSA function on which the underlying computational assumption holds.

Definition 4 (RSA Function) *An RSA generator $\mathsf{RSAKeyGen}$ with associated security parameter l is a randomized algorithm that takes no input and returns a pair $((e, N), (d, N))$ such that (1) p, q are distinct and odd primes with each being about $l/2$ bits long, (2) $N = pq$ where $2^{l-1} \leq N < 2^l$ and (3) $e, d \in \mathbb{Z}^\star_{\varphi(N)}$ are integers satisfying $ed \equiv 1 \bmod \varphi(N)$. We call N an RSA modulus, (e, N) the public key and (d, N) the private key. The RSA function is a family of functions $\mathsf{RSA}_{N,f} : \mathbb{Z}^\star_N \to \mathbb{Z}^\star_N$ defined by $\mathsf{RSA}_{N,f}(w) \equiv w^f \bmod N$ for all $w \in \mathbb{Z}^\star_N$. That is, the encryption function $\mathsf{RSA}_{N,e}$ is defined by $\mathsf{RSA}_{N,e}(x) = y \equiv x^e \bmod N$ and the decryption function $\mathsf{RSA}_{N,d}$ is $\mathsf{RSA}_{N,d}(y) = x \equiv y^d \bmod N$ both of which are permutations on $\mathbb{Z}^\star_N$ and inverses of each other.*

The computational assumption of the RSA function is equivalent to one-wayness (non-invertibility) of RSA: given $(N, e, y = \mathsf{RSA}_{N,e}(x))$ it is hard to compute x. Formally,

Definition 5 (One-wayness of RSA) *Suppose that the RSA function is defined by Definition 4 and an adversary $\mathcal{I}$ is given the RSA instance $(N, e, y = \mathsf{RSA}_{N,e}(x))$. The RSA function is said to be* one-way *if the success probability of $\mathcal{I}$, defined as*

$$\mathsf{Succ}^{\mathsf{ow}}_{RSA}(\mathcal{I}) = \Pr[x = x' | x' \leftarrow \mathcal{I}(N, e, y)], \tag{2}$$

is negligible in the security parameter l.

2 Our Protocol

Before presenting an efficient and leakage-resilient RSA-based AKE (for short, $\mathsf{RSA\text{-}AKE}$) protocol, we will start by giving a considered situation and some notations to be used.

2.1 Considered Situation

Consider the following situation where a client is communicating with many disparate servers[2]. In particular, we focus on unbalanced wireless networks where the client has easy-to-be-lost/stolen devices (e.g., mobile phones or PDAs) with very-restricted computing power but some memory capacity itself, on the other hand, each server has its database and enormous computing power enough to generate a pair of RSA keys and to perform the RSA decryption function. Here, we do not assume that each server is completely secure against possible attacks (e.g., virus, hackers or insider attacks). In addition, neither PKI nor TRM is available.

[2] For the sake of simplicity, we assign the servers consecutive integer $i \geq 1$ where $\mathcal{S}_i$ can be regarded as the i-th server.

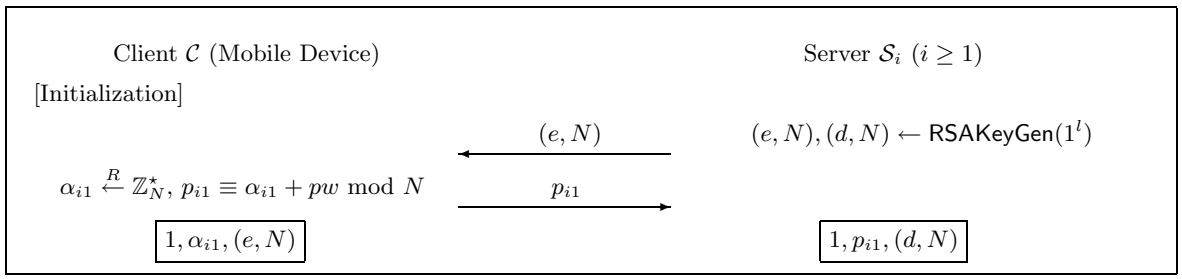

Fig. 1. The initialization of RSA-based AKE (RSA-AKE) protocol where the enclosed values in rectangle represent stored secrets of client and server, respectively

2.2 Notations

Let k and l denote the security parameters, where k can be thought of as the general security parameter for hash functions (say, 160 bits) and l ($l > k$) can be thought of as the security parameter for RSA (say, 1024 bits). Let D be a dictionary size of passwords (say, 36 bits for alphanumerical passwords with 6 characters). Let $\{0,1\}^\star$ denote the set of finite binary strings and $\{0,1\}^k$ the set of binary strings of length k. If A is a set, then $a \stackrel{R}{\leftarrow} A$ indicates the process of selecting a at random and uniformly over A. Let "$||$" denote the concatenation of bit strings in $\{0,1\}^\star$.

Let us define secure hash functions. While $\mathcal{G} : \{0,1\}^\star \rightarrow \mathbb{Z}_N^\star \backslash \{1\}$ denotes a full-domain hash (FDH) function, the other hash functions are denoted $\mathcal{H}_j : \{0,1\}^\star \rightarrow \{0,1\}^k$ for $j = 1, 2, 3$ and 4. Here $\mathcal{G}$ and $\mathcal{H}_j$ are distinct random functions one another. Let $\mathcal{C}$ and $\mathcal{S}$ be the identities of client and server, respectively, with representing each ID $\in \{0,1\}^\star$ as well.

2.3 The RSA-AKE Protocol

Here we propose an efficient and leakage-resilient RSA-based AKE (RSA-AKE) protocol suitable for the above-mentioned situation. The whole protocol is illustrated in Fig. 1 and 2.

Initialization During the initialization phase, client $\mathcal{C}$ registers a verification data to one of the different servers $\mathcal{S}_i$ $(i \geq 1)$. At first, server $\mathcal{S}_i$ sends its RSA public key (e, N), which is generated from $\mathsf{RSAKeyGen}(1^l)$, to the client. The latter picks a secret value α_{i1} randomly chosen in $\mathbb{Z}_N^\star$ and registers securely a verification data p_{i1} to server $\mathcal{S}_i$:

$$p_{i1} \equiv \alpha_{i1} + \alpha_0 \bmod N \tag{3}$$

and sets the term $\alpha_0 = pw$ where pw is the client's password[3]. Since both α_{i1} and p_{i1} are in the set of the same length, each is a share of $(2,2)$-threshold secret sharing scheme for α_0 [7].

Then client $\mathcal{C}$ remembers his password pw *and* additionally stores the secret value α_{i1} and the RSA public key (e, N) on insecure devices (e.g., mobile devices or smart cards) which may happen to leak α_{i1} and (e, N). The server $\mathcal{S}_i$ also stores the verification data p_{i1} and its RSA private key (d, N) on its databases both of which may be leaked out. Finally, they set a counter j as 1. Note that this initialization is done only once.

***j*-th Protocol Execution** When client $\mathcal{C}$ wants to share an authenticated session key securely with server $\mathcal{S}_i$, they run the j-th $(j \geq 1)$ execution of the RSA-AKE protocol as follows. At the start of the j-th protocol execution, client $\mathcal{C}$ and server $\mathcal{S}_i$ hold $(j, \alpha_{ij}, (e, N))$ and $(j, p_{ij}, (d, N))$, respectively, where $p_{ij} \equiv \alpha_{ij} + pw \bmod N$. The client $\mathcal{C}$ should recover the verification data p_{ij} by adding the secret value α_{ij} stored on devices with the password pw remembered in his mind. Then the client chooses a random value x from $\mathbb{Z}_N^\star$ and sends $(\mathcal{C}, j, z)$ to server $\mathcal{S}_i$ after calculating z using a mask generation function as the product of an encryption of x under the RSA public key (e, N) with a full-domain hash of (j, p_{ij}). If the received counter j is correct, the server divides z by a hash of the counter and its verification data

[3] The password pw is drawn from password space $\mathbb{D}_{\mathsf{Password}}$ according to a certain probability distribution.

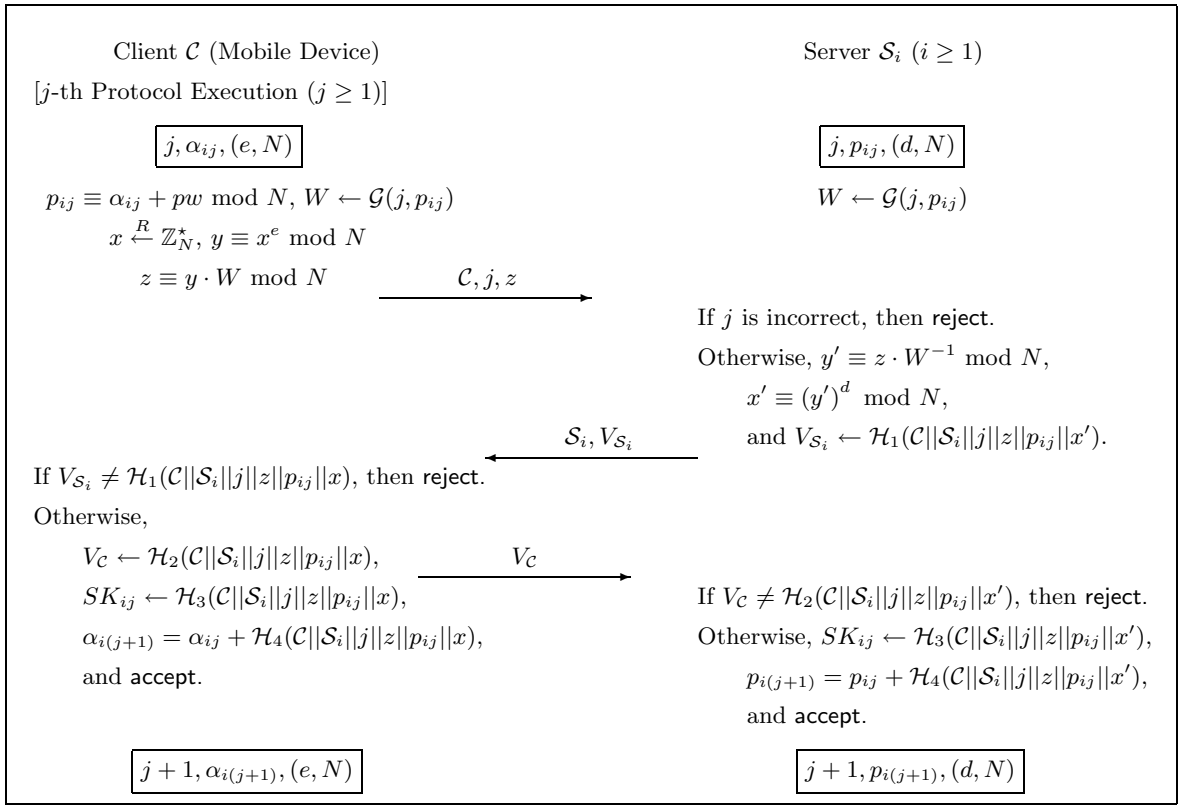

Fig. 2. The j-th protocol execution of RSA-AKE protocol where the enclosed values in rectangle represent stored secrets of client and server, respectively

p_{ij}, and then decrypts the resultant value under its RSA private key (d, N) so as to obtain x. Then, server $\mathcal{S}_i$ computes and sends its authenticator $V_{\mathcal{S}_i}$ to client $\mathcal{C}$.

Upon receiving $(\mathcal{S}_i, V_{\mathcal{S}_i})$ from the server, client $\mathcal{C}$ computes his authenticator $V_{\mathcal{C}}$ and a session key SK_{ij}, as long as $\mathcal{H}_1(\mathcal{C}||\mathcal{S}_i||j||z||p_{ij}||x)$ is equal to $V_{\mathcal{S}_i}$, and sends $V_{\mathcal{C}}$ to server $\mathcal{S}_i$. If the authenticator $V_{\mathcal{C}}$ is valid, server $\mathcal{S}_i$ actually computes a session key SK_{ij} that will be used for their subsequent cryptographic algorithms (e.g., AES or HMAC).

At the end of the j-th protocol execution, client $\mathcal{C}$ refreshes the secret value α_{ij} to a new one $\alpha_{i(j+1)}$ without changing his password[4]. In the same way, server $\mathcal{S}_i$ also refreshes the verification data p_{ij} to a new one $p_{i(j+1)}$. Finally, client $\mathcal{C}$ stores $\big(j+1, \alpha_{i(j+1)}, (e, N)\big)$ on his devices and server $\mathcal{S}_i$ stores $\big(j+1, p_{i(j+1)}, (d, N)\big)$ on its databases for the next session.

3 Security

In this section we show the RSA-AKE protocol of Fig. 2. is provably secure in the random oracle model [2][5], under the assumption that inverting an RSA instance is hard, by Definition 2.

3.1 Security Proof

In order to simplify the security proof, we omit the index i and only consider the first two flows of the j-th protocol execution (unilateral authentication of $\mathcal{S}$ to $\mathcal{C}$). The latter is due to the well-known fact that the basic approach in the literature for adding authentication to an AKE protocol is to use the

[4] Notice that the frequent change of passwords might incur the risk of password to be exposed, simply because people tends to write it down on somewhere or needs considerable efforts to remember new passwords.

[5] Note that security in the random oracle model is only a heuristic: it does not imply security in the real world [4]. Nevertheless, the random oracle model is a useful tool for validating natural cryptographic constructions. Security proofs in this model prove security against adversaries that are confined to the random oracle world.

– For a hash-query $\mathcal{H}_i(q)$ (resp., $\mathcal{H}'_i(q)$), such that a record (i, q, r) appears in $\Lambda_{\mathcal{H}_i}$ (resp., $\Lambda_{\mathcal{H}'_i}$), the answer is r. Otherwise one chooses a random element $r \stackrel{R}{\leftarrow} \{0,1\}^{k_i}$, answers with it, and adds the record (i, q, r) to $\Lambda_{\mathcal{H}_i}$ (resp., $\Lambda_{\mathcal{H}'_i}$).
  ▶ **Rule** $\mathcal{H}^{(1)}$
  Nothing to do. % To be defined later
– For a hash-query $\mathcal{G}(j, q)$, such that a record $(j, q, r, \star, \star)$ appears in $\Lambda_{\mathcal{G}}$, the answer is r. Otherwise the answer r is defined according to the following rule:
  ▶ **Rule** $\mathcal{G}^{(1)}$
  Choose a random element $r \stackrel{R}{\leftarrow} \mathbb{Z}_N^\star$. The record $(j, q, r, \perp, \perp)$ is added to $\Lambda_{\mathcal{G}}$.
  Note: the fourth and fifth components of the elements of this list will be explained later.

Fig. 3. Simulation of the hash functions: $\mathcal{G}$ and $\mathcal{H}_i$ oracles

distributed Diffie-Hellman key or the shared secret to construct a simple "authenticator" for the other party [1, 5]. Therefore, the security proof with unilateral authentication can be extended to one with mutual authentication by simply adding the authenticator of $\mathcal{C}$ (the third flow) as in Fig. 2. However, this makes a proof more complicated.

Here we assert that the RSA-AKE protocol distributes semantically-secure session keys and provides unilateral authentication for the server $\mathcal{S}$.

Theorem 1 (LR-AKE/UA Security) *Let P be the* RSA-AKE *protocol of Fig. 2., where passwords are chosen from a dictionary of size D and the client's stored secret (i.e., $\alpha_{ij}, (e, N)$) is provided through the* Leak*-query. For any adversary $\mathcal{A}$ within a polynomial time t, with less than q_s active interactions with the parties (*Send*-queries), q_p passive eavesdroppings (*Execute*-queries), and asking q_g and q_h hash queries to $\mathcal{G}$ and any $\mathcal{H}_i$ respectively, $\mathsf{Adv}_P^{\mathsf{lr-ake}}(\mathcal{A}) \leq 4\varepsilon$ and $\mathsf{Adv}_P^{\mathsf{S-auth}}(\mathcal{A}) \leq \varepsilon$, with ε upper-bounded by*

$$\frac{3q_{\mathcal{S}}}{D} + 10\mathsf{Succ}^{\mathsf{ow}}_{RSA}\left(q_h^2, t + 2q_h^2\tau_{law}\right) + \frac{q_{\mathcal{C}}}{2^{k_1}} + \frac{6q_{\mathcal{S}} + (q_{\mathcal{C}} + q_p)^2 + (q_g + q_h)^2}{2^{l+1}} \ , \tag{4}$$

where $q_{\mathcal{C}}$ and $q_{\mathcal{S}}$ denote the number of $\mathcal{C}$ and $\mathcal{S}$ instances involved during the attack (each upper-bounded by $q_p + q_s$), k_1 is the output length of $\mathcal{H}_1$, l is the security parameter, and τ_{law} is the computational time needed for modular multiplication or modular division.

Informally speaking, an adversary who knows the client's stored secret cannot determine the correct password through off-line dictionary attacks since generating the valid authenticator after computing z falls into on-line dictionary attacks (which can be easily prevented and detected).

Proof 1. In this proof, we incrementally define a sequence of games starting at the real game $\mathbf{G}_0$ and ending up at $\mathbf{G}_5$. We use Shoup's lemma [8] to bound the probability of each event in these games.

Game $\mathbf{G}_0$: This is the real protocol in the random oracle model. We are interested in the following two events:
– S_0 (for semantic security) which occurs if the adversary correctly guesses the bit b involved in the Test-query;
– A_0 (for $\mathcal{S}$-authentication) which occurs if an instance $\mathcal{C}^I$ accepts with no partner instance $\mathcal{S}^J$ with the same transcript $((\mathcal{C}, j, z), (\mathcal{S}, V_{\mathcal{S}}))$

$$\mathsf{Adv}_P^{\mathsf{lr-ake}}(\mathcal{A}) = 2\Pr[\mathsf{S}_0] - 1 \ , \mathsf{Adv}_P^{\mathsf{S-auth}}(\mathcal{A}) = \Pr[\mathsf{A}_0] \ . \tag{5}$$

In any game $\mathbf{G}_n$ below, we study the event A_n and the restricted event $\mathsf{SwA}_n = \mathsf{S}_n \wedge \neg\mathsf{A}_n$.

Game $\mathbf{G}_1$: In this game, we simulate the hash oracles ($\mathcal{G}, \mathcal{H}_1, \mathcal{H}_3$ and $\mathcal{H}_4$, but as well additional hash functions $\mathcal{H}'_i : \{0,1\}^\star \to \{0,1\}^{k_i}$ (for $i = 1, 3, 4$) that will appear in the Game $\mathbf{G}_3$) as usual by maintaining hash lists $\Lambda_{\mathcal{G}}$, $\Lambda_{\mathcal{H}_i}$ and $\Lambda_{\mathcal{H}'_i}$ (see Fig. 3). We also simulate all the instances, as the real parties would do, for the Send, Execute, Reveal, Leak and Test-queries (see Fig. 4). From this simulation, we can easily see that the game is perfectly indistinguishable from the real attack.

Send-queries to $\mathcal{C}$

We answer to the Send-queries to a $\mathcal{C}$-instance as follows:

- A $\mathsf{Send}(\mathcal{C}^I, \mathtt{Start})$-query is processed according to the following rules:
  - ▶ **Rule C1**$^{(1)}$
    Compute $p_j \equiv \alpha_j + pw \bmod N$.
  - ▶ **Rule C2**$^{(1)}$
    Generate $(x, y \equiv x^e \bmod N)$, and compute $W \leftarrow \mathcal{G}(j, p_j)$ and $z \equiv y \times W \bmod N$.

  Then the query is answered with $(\mathcal{C}, j, z)$, and the instance goes to an expecting state.
- If the instance $\mathcal{C}^I$ is in an expecting state, a query $\mathsf{Send}(\mathcal{C}^I, (\mathcal{S}, V_\mathcal{S}))$ is processed by computing the alleged authenticator, the session key and the refreshed secret. We apply the following rules.
  - ▶ **Rule C3**$^{(1)}$
    Compute the expected authenticator and the session key
    $V'_\mathcal{S} \leftarrow \mathcal{H}_1(\mathcal{C}||\mathcal{S}||j||z||p_j||x)$, $SK_\mathcal{C} \leftarrow \mathcal{H}_3(\mathcal{C}||\mathcal{S}||j||z||p_j||x)$.
  - ▶ **Rule C4**$^{(1)}$
    Compute the refreshed secret $\alpha_{j+1} = \alpha_j + \mathcal{H}_4(\mathcal{C}||\mathcal{S}||j||z||p_j||x)$.

  If $V'_\mathcal{S} = V_\mathcal{S}$, the instance accepts. In any case, it terminates.

Send-queries to $\mathcal{S}$

We answer to the Send-queries to a $\mathcal{S}$-instance as follows:

- A $\mathsf{Send}(\mathcal{S}^J, (\mathcal{C}, j, z))$-query is first processed by checking that j is the correct counter, and in the case of correct j it is processed by computing the authenticator, the session key and the refreshed secret. We apply the following rules:
  - ▶ **Rule S1**$^{(1)}$
    Compute $W \leftarrow \mathcal{G}(j, p_j), y' \equiv z \times W^{-1} \bmod N$ and $x' \equiv (y')^d \bmod N$.
  - ▶ **Rule S2**$^{(1)}$
    Compute the authenticator and the session key
    $V_\mathcal{S} \leftarrow \mathcal{H}_1(\mathcal{C}||\mathcal{S}||j||z||p_j||x')$, $SK_\mathcal{S} \leftarrow \mathcal{H}_3(\mathcal{C}||\mathcal{S}||j||z||p_j||x')$.
  - ▶ **Rule S3**$^{(1)}$
    Compute the refreshed secret $p_{j+1} = p_j + \mathcal{H}_4(\mathcal{C}||\mathcal{S}||j||z||p_j||x')$.

  Finally the instance accepts, and the query is answered with $(\mathcal{S}, V_\mathcal{S})$.

Other queries

- An $\mathsf{Execute}(\mathcal{C}^I, \mathcal{S}^J)$-query is processed using successively the above simulations of the Send-queries: $(\mathcal{C}, j, z) \leftarrow \mathsf{Send}(\mathcal{C}^I, \mathtt{Start})$, $(\mathcal{S}, V_\mathcal{S}) \leftarrow \mathsf{Send}(\mathcal{S}^J, (\mathcal{C}, j, z))$, $\mathsf{Send}(\mathcal{C}^I, (\mathcal{S}, V_\mathcal{S}))$, and then outputting the transcript $((\mathcal{C}, j, z), (\mathcal{S}, V_\mathcal{S}))$.
- A $\mathsf{Reveal}(U)$-query returns the session key ($SK_\mathcal{C}$ or $SK_\mathcal{S}$) computed by the instance U (if the latter has accepted).
- A $\mathsf{Leak}(\mathcal{C}^I)$-query returns the stored secret $(\alpha_j, (e, N))$ by the instance $\mathcal{C}^I$.
- A $\mathsf{Test}(U)$-query first gets SK from $\mathsf{Reveal}(U)$, and flip a coin b. If $b = 1$, we return the value of the session key SK, otherwise we return a random value drawn from $\{0,1\}^{k_3}$.

Fig. 4. Simulation of the RSA-AKE protocol

Game $\mathbf{G_2}$: For an easier analysis in the following, we first forward any query to $\mathcal{H}_i$ to $\mathcal{G}$:

- ▶ **Rule $\mathcal{H}^{(2)}$**
  The query q is parsed as $q = \mathcal{C}||\mathcal{S}||j||z||p_j||x$, then one queries $\mathcal{G}(j, p_j)$.

The number of queries to $\mathcal{G}$ thus becomes $q'_g \leq q_g + q_h$. Furthermore we exclude games in which some events (Coll_2) are unlikely to happen:

- collision of the partial transcript $(\mathcal{C}, j, z)$: any adversary tries to find out at least one pair (j, z), coinciding with the challenge transcript, involving the honest party $\mathcal{C}$, and then obtain the corresponding session key (i.e., the same as the challenge session key) using the Reveal-query;
- collision on the output of $\mathcal{G}$.

Both probabilities are bounded by the birthday paradox:

$$\Pr[\mathsf{Coll}_2] \leq \frac{(q_\mathcal{C} + q_p)^2 + q'^2_g}{2^{l+1}} \ . \tag{6}$$

Game $\mathbf{G_3}$: In order to make the authenticators and the session keys unpredictable to any adversary, we compute them using the private oracles $\mathcal{H}'_1$ and $\mathcal{H}'_3$ (instead of $\mathcal{H}_1$ and $\mathcal{H}_3$), respectively, so that the values are completely independent from the random oracles as well as $\mathcal{G}$. We reach this aim by using the following rules:

▶ **Rule C3/S2**$^{(3)}$
Compute the authenticator
$V_{\mathcal{S}} \leftarrow \mathcal{H}'_1(\mathcal{C}||\mathcal{S}||j||z)$.
Compute the session key
$SK_{\mathcal{C}/\mathcal{S}} \leftarrow \mathcal{H}'_3(\mathcal{C}||\mathcal{S}||j||z)$.

We also use the private oracle $\mathcal{H}'_4$ (instead of $\mathcal{H}_4$) so that the refreshed secrets are unpredictable to any adversary as well. We modify the simulation by using the following rules:

▶ **Rule C1**$^{(3)}$
Choose two random elements $(\beta_j, \gamma_j) \stackrel{R}{\leftarrow} (\mathbb{Z}_N^\star)^2$, and set $\alpha_j \leftarrow \beta_j$ and $p_j \leftarrow \gamma_j$.

▶ **Rule C4/S3**$^{(3)}$
Compute $\alpha_{j+1} = \alpha_j + \mathcal{H}'_4(\mathcal{C}||\mathcal{S}||j||z)$ and $p_{j+1} = p_j + \mathcal{H}'_4(\mathcal{C}||\mathcal{S}||j||z)$.

Since the secret $x = x' = \mathsf{RSA}_{N,d}\left(z \times W^{-1}\right)$ depends on $(e, N), z$ and the parties (the common secret p_j), the games $\mathbf{G}_3$ and $\mathbf{G}_2$ are indistinguishable unless some specific hash queries are asked, denoted by event $\mathsf{AskH}_3 = \mathsf{AskH1}_3 \vee \mathsf{AskH3w1}_3 \vee \mathsf{AskH4w13}_3$, where $W = \mathcal{G}(j, p_j)$:

- $\mathsf{AskH1}_3$: $\left(\mathcal{C}||\mathcal{S}||j||z||p_j||\mathsf{RSA}_{N,d}\left(z \times W^{-1}\right)\right)$ has been queried by $\mathcal{A}$ to $\mathcal{H}_1$ for some transcript $((\mathcal{C}, j, z), (\mathcal{S}, V_{\mathcal{S}}))$;
- $\mathsf{AskH3w1}_3$: $\left(\mathcal{C}||\mathcal{S}||j||z||p_j||\mathsf{RSA}_{N,d}\left(z \times W^{-1}\right)\right)$ has been queried by $\mathcal{A}$ to $\mathcal{H}_3$ for some transcript $((\mathcal{C}, j, z), (\mathcal{S}, V_{\mathcal{S}}))$, where some party has accepted, but event $\mathsf{AskH1}_3$ did not happen;
- $\mathsf{AskH4w13}_3$: $\left(\mathcal{C}||\mathcal{S}||j||z||p_j||\mathsf{RSA}_{N,d}\left(z \times W^{-1}\right)\right)$ has been queried by $\mathcal{A}$ to $\mathcal{H}_4$ for some transcript $((\mathcal{C}, j, z), (\mathcal{S}, V_{\mathcal{S}}))$, where some party has accepted, but both $\mathsf{AskH1}_3$ and $\mathsf{AskH3w1}_3$ events did not happen.

After this modification, we do no longer need to know the value x nor to compute the value x' either. Thus we can simplify the corresponding rules:

▶ **Rule C2**$^{(3)}$
Generate a random pair $(\hat{x}, z = \mathsf{RSA}_{N,e}(\hat{x}))$.

▶ **Rule S1**$^{(3)}$
Do nothing.

Finally, one can notice that the actual password is not used any more. By the isomorphic property of $\mathsf{RSA}_{N,f}$ from $\mathbb{Z}_N^\star$ to $\mathbb{Z}_N^\star$, the new value z is perfectly indistinguishable from before, since there exists a unique pair (x, y),

$$x = \mathsf{RSA}_{N,d}\left(z \times W^{-1}\right) = \hat{x} \times \mathsf{RSA}_{N,d}\left(W^{-1}\right) \qquad y = \mathsf{RSA}_{N,e}(x) \qquad \text{such that } z = y \times W.$$

The authenticator is computed with a random oracle that is private to the simulator, then one can remark that it cannot be guessed by the adversary, better than at random for each attempt, unless the same partial transcript (j, z) appeared in another session with a real instance $\mathcal{C}^I$. But such a case has already been excluded (in Game $\mathbf{G}_2$). Similarly, this can be applied to the session key.

$$\Pr[\mathsf{A}_3] \leq \frac{q_{\mathcal{C}}}{2^{k_1}} \qquad \Pr[\mathsf{SwA}_3] = \frac{1}{2}\ . \tag{7}$$

Since collision of the partial transcript has been excluded, the event $\mathsf{AskH1}$ can be split in three disjoint sub-cases:

- $\mathsf{AskH1\text{-}Passive}_3$: the transcript $((\mathcal{C}, j, z), (\mathcal{S}, V_{\mathcal{S}}))$ comes from an execution between instances of $\mathcal{C}$ and $\mathcal{S}$ ($\mathsf{Execute}$-queries or forward of Send-queries, relay of part of them). In this case both (j, z) and the RSA key pair have been simulated where the latter has been provided through the Leak-query;
- $\mathsf{AskH1\text{-}WithS}_3$: the execution involved an instance of $\mathcal{S}$, but (j, z) has not been sent by any instance of $\mathcal{C}$. This means that the RSA key pair has been simulated, but (j, z) has been produced by the adversary;
- $\mathsf{AskH1\text{-}WithC}_3$: the execution involved an instance of $\mathcal{C}$, but an instance of $\mathcal{S}$ has not been involved during the attack. This means that both (j, z) and the RSA key pair have been simulated.

Game $\mathbf{G}_4$: In order to evaluate the above events, we introduce a random RSA instance (N, e, ρ), where ρ is a uniformly distributed random element in $\mathbb{Z}_N^\star$ (or equivalently, because of the isomorphism property, σ is randomly drawn from $\mathbb{Z}_N^\star$ and $\rho = \mathsf{RSA}_{N,e}(\sigma)$). We are looking for the value σ.
We introduce the instance (N, e, ρ) in the simulation of the oracle $\mathcal{G}$, using again the homomorphic property of RSA. Specifically, the simulation introduces values in the third and fourth elements of $\Lambda_{\mathcal{G}}$: the pre-image of the answer by $\mathsf{RSA}_{N,e}$, but does not use it. We modify the simulation as

follows:

► **Rule** $\mathcal{G}^{(4)}$

Generate a random pair $(u, v = \mathsf{RSA}_{N,e}(u))$, and with a random bit b compute $r \leftarrow v$, if $b = 0$, and $r \leftarrow v \cdot \rho$, if $b = 1$. Then, the record (j, q, r, u, b) is added to $\Lambda_{\mathcal{G}}$.

The probability remains unchanged because of the homomorphic property.

Game $\mathbf{G}_5$: It is now possible to evaluate the probability of the event AskH (or more precisely, the sub-cases). First, one can see that the password is never used during the simulation of $\mathbf{G}_4$. It doest not need to be chosen in advance, but at the very end only. Then, an information-theoretic analysis can be done which simply uses cardinalities of some sets. This is crucial that the entire simulation is basically independent from the chosen password.

To this aim, we first exclude a few more games, wherein for some pair (j, z), involved in a communication between an instance $\mathcal{S}^J$ and either the adversary or an instance $\mathcal{C}^I$, there exist two distinct values p_j, and thus elements W, since $W = \mathcal{G}(j, p_j)$ such that both the tuples $(j, z, p_j, \mathsf{RSA}_{N,d}(z/W))$ are in $\Lambda_{\mathcal{H}}$ (which event is denoted CollH_5):

$$|\Pr[\mathsf{AskH}_5] - \Pr[\mathsf{AskH}_4]| \leq \Pr[\mathsf{CollH}_5] \ . \tag{8}$$

With the following lemma, the event CollH_5 can be upper-bounded.

Lemma 1 *For any pair (j, z) involved in a communication with an instance $\mathcal{S}^J$, unless one can invert the RSA instance, there is at most one valid element W obtained from $\mathcal{G}$ such that $(j, z, p_j, \mathsf{RSA}_{N,d}(z/W))$ in $\Lambda_{\mathcal{H}}$:*

$$\Pr[\mathsf{CollH}_5] \leq 2\mathsf{Succ}^{\mathsf{ow}}_{RSA}(q_h^2, t + 2q_h^2\tau_{law}) \ . \tag{9}$$

Proof. We show the proof by contradiction. Here we assume that there exists (j, z) involved in a communication, and $W_0 = \mathsf{RSA}_{N,e}(u_0) \cdot b_0 \cdot \rho$ and $W_1 = \mathsf{RSA}_{N,e}(u_1) \cdot b_1 \cdot \rho$, both obtained from the $\mathcal{G}$ oracle, such that the tuples $(j, z, p_{ji}, \mathsf{RSA}_{N,d}(z/W_i))$ are in $\Lambda_{\mathcal{H}}$, for $i = 0, 1$. Then,

$$Z \stackrel{\mathsf{def}}{=} \frac{\mathsf{RSA}_{N,d}(z/W_1)}{\mathsf{RSA}_{N,d}(z/W_0)} = \mathsf{RSA}_{N,d}(W_0/W_1) \ . \tag{10}$$

With probability of $1/2$, the bits b_0 and b_1 involved in W_0 and W_1 are distinct. Without loss of generality, we may assume that $b_i = i$ for $i = 0, 1$:

$$Z = \mathsf{RSA}_{N,d}\left(\frac{\mathsf{RSA}_{N,e}(u_0)}{\mathsf{RSA}_{N,e}(u_1) \cdot \rho}\right) = \frac{u_0}{u_1 \cdot \mathsf{RSA}_{N,d}(\rho)} = \frac{u_0}{u_1 \cdot \sigma} \ . \tag{11}$$

As a consequence, $\sigma = u_0/(u_1 \cdot Z)$. By either guessing the two queries asked to the $\mathcal{H}_i$ or checking for each pair the validity of the computed σ, one concludes the proof. □

In order to conclude the proof, let us study separately the three sub-cases of $\mathsf{AskH1}$, and then $\mathsf{AskH3w1}$ and $\mathsf{AskH4w13}$ (keeping in mind the absence of several kinds of collisions: for partial transcripts, for $\mathcal{G}$, and for p_j in $\mathcal{H}$-queries):

– $\mathsf{AskH1\text{-}Passive}$: about the passive transcripts (in which both (j, z) and the RSA key pair have been simulated), one can state the following lemma:

Lemma 2 *For any pair (j, z) involved in a passive transcript, unless one can invert the RSA instance, there is no valid element W such that $(j, z, p_j, \mathsf{RSA}_{N,d}\ (z/W))$ in $\Lambda_{\mathcal{H}}$:*

$$\Pr[\mathsf{AskH1\text{-}Passive}_5] \leq 2\mathsf{Succ}^{\mathsf{ow}}_{RSA}(q_h, t + 2q_h\tau_{law}) \ . \tag{12}$$

Proof. Assume that there exist (j, z) involved in a passive transcript and $W = \mathsf{RSA}_{N,e}(u) \cdot b \cdot \rho$ such that the tuple $(j, z = \mathsf{RSA}_{N,e}(\hat{x}), p_j, Z \stackrel{\mathsf{def}}{=} \mathsf{RSA}_{N,d}(z/W))$ is in $\Lambda_{\mathcal{H}}$. Then, as above, with probability of $1/2$, $b = 1$:

$$Z = \mathsf{RSA}_{N,d}\left(\frac{\mathsf{RSA}_{N,e}(\hat{x})}{\mathsf{RSA}_{N,e}(u) \cdot \rho}\right) = \frac{\hat{x}}{u \cdot \mathsf{RSA}_{N,d}(\rho)} = \frac{\hat{x}}{u \cdot \sigma} \ . \tag{13}$$

As a consequence, $\sigma = \hat{x}/(u \cdot Z)$. By either guessing the query asked to the $\mathcal{H}_i$ or checking the validity of σ for each candidate, one concludes the proof. □

– AskH1-WithS: the above Lemma 1, applied to games where the event CollH_5 did not happen (and without $\mathcal{G}$-collision), states that for each pair (j, z) involved in a transcript with an instance $\mathcal{S}^J$, there is at most one element p_j such that for $W = \mathcal{G}(j, p_j)$, the corresponding tuple is in $\Lambda_{\mathcal{H}}$. Thus, the probability for the adversary (who may have obtained α_j in the Leak-query, denoted by event Leak_5) over a random password is upper-bounded by:

$$\Pr[\mathsf{AskH1\text{-}WithS}_5] \leq \frac{q_{\mathcal{S}}}{2^l} + \frac{q_{\mathcal{S}}}{D} \ . \tag{14}$$

– AskH1-WithC: this may correspond to an attack where the adversary tries to impersonate $\mathcal{S}$ to $\mathcal{C}$ (break unilateral authentication). But each authenticator sent by the adversary has been computed with at most one p_j. Thus, the above Lemma 2 also applies to this case:

$$\Pr[\mathsf{AskH1\text{-}WichC}_5] \leq 2\mathsf{Succ}^{\mathsf{ow}}_{RSA}(q_h, t + 2q_h\tau_{law}) \ . \tag{15}$$

About AskH3w1 (when the above three events did not happen), it means that only executions with an instance of $\mathcal{S}$ may lead to acceptance (and either $\mathcal{C}$ or the adversary). Exactly the same analysis as for AskH1-Passive and AskH1-WithS leads to

$$\Pr[\mathsf{AskH3w1}_5] \leq \frac{q_{\mathcal{S}}}{2^l} + \frac{q_{\mathcal{S}}}{D} + 2\mathsf{Succ}^{\mathsf{ow}}_{RSA}(q_h, t + 2q_h\tau_{law}) \ . \tag{16}$$

About AskH4w13 (when both of the events AskH1 and AskH3w1 did not happen), the same analysis as for AskH3w1 leads to

$$\Pr[\mathsf{AskH4w13}_5] \leq \frac{q_{\mathcal{S}}}{2^l} + \frac{q_{\mathcal{S}}}{D} + 2\mathsf{Succ}^{\mathsf{ow}}_{RSA}(q_h, t + 2q_h\tau_{law}) \ . \tag{17}$$

As a conclusion, we get an upper-bound for the probability of AskH_5 by combining all the cases:

$$\Pr[\mathsf{AskH}_5] \leq \frac{3q_{\mathcal{S}}}{2^l} + \frac{3q_{\mathcal{S}}}{D} + 8\mathsf{Succ}^{\mathsf{ow}}_{RSA}(q_h, t + 2q_h\tau_{law}) \ . \tag{18}$$

Combining equations (6), (7), (9) and (18), one gets either

$$\Pr[\mathsf{A}_0] \leq \frac{q_{\mathcal{C}}}{2^{k_1}} + \Delta \qquad \Pr[\mathsf{SwA}_0] = \frac{1}{2} + \Delta \ , \tag{19}$$

where

$$\begin{aligned} \Delta &\leq 2\mathsf{Succ}^{\mathsf{ow}}_{RSA}\left(q_h^2, t + 2q_h^2\tau_{law}\right) + 8\mathsf{Succ}^{\mathsf{ow}}_{RSA}\left(q_h, t + 2q_h\tau_{law}\right) + \frac{3q_{\mathcal{S}}}{D} + \frac{3q_{\mathcal{S}}}{2^l} + \frac{(q_{\mathcal{C}} + q_p)^2 + q_g'^2}{2^{l+1}} \\ &\leq \frac{3q_{\mathcal{S}}}{D} + \frac{6q_{\mathcal{S}} + (q_{\mathcal{C}} + q_p)^2 + q_g'^2}{2^{l+1}} + 10\mathsf{Succ}^{\mathsf{ow}}_{RSA}\left(q_h^2, t + 2q_h^2\tau_{law}\right) \ . \end{aligned} \tag{20}$$

One can get the result as desired by noting that $\Pr[\mathsf{S}_0] \leq \Pr[\mathsf{SwA}_0] + \Pr[\mathsf{A}_0]$.

Theorem 2 (LR-AKE/UA Security) *Let P be the* RSA-AKE *protocol of Fig. 2., where the server's stored secret (i.e., (d, N)) is provided through the* Leak*-query. For any adversary $\mathcal{A}$, with less than q_s active interactions with the parties (*Send*-queries), q_p passive eavesdroppings (*Execute*-queries), and asking q_g and q_h hash queries to $\mathcal{G}$ and any $\mathcal{H}_i$ respectively, $\mathsf{Adv}_P^{\mathsf{lr-ake}}(\mathcal{A}) \leq 4\varepsilon$ and $\mathsf{Adv}_P^{\mathsf{S-auth}}(\mathcal{A}) \leq \varepsilon$, with ε upper-bounded by*

$$\frac{q_{\mathcal{C}}}{2^{k_1}} + \frac{6(q_p + q_{\mathcal{S}}) + 3(q_{\mathcal{C}} + q_p)^2 + 2(q_g + q_h)^2}{2^{l+1}} \ , \tag{21}$$

where $q_{\mathcal{C}}$ and $q_{\mathcal{S}}$ denote the number of $\mathcal{C}$ and $\mathcal{S}$ instances involved during the attack (each upper-bounded by $q_p + q_s$), k_1 is the output length of $\mathcal{H}_1$ and l is the security parameter.

Informally speaking, an adversary who knows the server's RSA private key cannot perform even on-line dictionary attacks since the authentication depends on the strong secret p_j.

Proof. In this proof, we incrementally define a sequence of games starting at the real game $\mathbf{G}_0$ and ending up at $\mathbf{G}_4$. Here we describe differences only from Proof 1.

Game $\mathbf{G}_0$: This is the same as $\mathbf{G}_0$ of Proof 1.

Game $\mathbf{G}_1$: In this game, we modify the simulations of the hash oracle $\mathcal{G}$ and the Leak query in Fig. 3 and 4, respectively, as follows:

- For a hash-query $\mathcal{G}(j, q)$, such that a record (j, q, r) appears in $\Lambda_{\mathcal{G}}$, the answer is r. Otherwise one chooses a random element $r \stackrel{R}{\leftarrow} \mathbb{Z}_N^\star$, answers with it, and adds the record (j, q, r) to $\Lambda_{\mathcal{G}}$;
- A $\mathsf{Leak}(\mathcal{S}^J)$-query returns the stored secret (d, N) by the instance $\mathcal{S}^J$.

The remaining is the same as $\mathbf{G}_1$ of Proof 1. From this simulation, we can easily see that the game is perfectly indistinguishable from the real attack.

Game $\mathbf{G}_2$: This is the same as $\mathbf{G}_2$ of Proof 1.

Game $\mathbf{G}_3$: This is the same as $\mathbf{G}_3$ of Proof 1 except the sub-cases (AskH1-WithS and AskH1-WithC) of AskH1:

- $\mathsf{AskH1\text{-}WithS}_3$: the execution involved an instance of $\mathcal{S}$, but (j, z) has not been sent by any instance of $\mathcal{C}$. This means that the RSA key pair has been simulated and provided through the Leak-query, but (j, z) has been produced by the adversary;
- $\mathsf{AskH1\text{-}WithC}_3$: the execution involved an instance of $\mathcal{C}$, but an instance of $\mathcal{S}$ has not been involved during the attack. This means that both (j, z) and the RSA key pair have been simulated where the latter has been provided through the Leak-query.

Game $\mathbf{G}_4$: Actually, we don't need to introduce an RSA instance in the simulation of the oracle $\mathcal{G}$ since the adversary knows the RSA private key through the Leak-query.
We first exclude an event (denoted by CollH_4), wherein for some pair (j, z) there exist two distinct values p_j, and thus elements W, since $W = \mathcal{G}(j, p_j)$ such that both the tuples $(j, z, p_j, \mathsf{RSA}_{N,d}(z/W))$ are in $\Lambda_{\mathcal{H}}$. As a result, the probability of CollH_4 is bounded by the birthday paradox:

$$\Pr[\mathsf{CollH}_4] \leq \frac{q_h^2}{2^{l+1}} \ . \tag{22}$$

This implies that for each pair (j, z) there is at most one element p_j such that, for $W = \mathcal{G}(j, p_j)$, the corresponding tuple is in $\Lambda_{\mathcal{H}}$.
In order to conclude the proof, let us evaluate separately the three sub-cases of AskH1, and then AskH3w1 and AskH4w13 (keeping in mind the absence of several kinds of collisions: for partial transcripts, for $\mathcal{G}$, and for p_j in $\mathcal{H}$-queries):

- AskH1-Passive: about the passive transcripts (in which both (j, z) and the RSA key pair have been simulated), the probability for the adversary over a random p_j is upper-bounded by:

$$\Pr[\mathsf{AskH1\text{-}Passive}_4] \leq \frac{q_p}{2^l} \ . \tag{23}$$

- AskH1-WithS: this may correspond to an attack where the adversary tries to impersonate $\mathcal{C}$ to $\mathcal{S}$. But each z sent by the adversary has been computed with at most one p_j. Thus,

$$\Pr[\mathsf{AskH1\text{-}WithS}_4] \leq \frac{q_{\mathcal{S}}}{2^l} \ . \tag{24}$$

- AskH1-WithC: this may correspond to an attack where the adversary tries to impersonate $\mathcal{S}$ to $\mathcal{C}$ (break unilateral authentication). But each authenticator sent by the adversary has been computed with at most one p_j. Thus,

$$\Pr[\mathsf{AskH1\text{-}WichC}_4]\frac{q_{\mathcal{C}}}{2^l} \ . \tag{25}$$

About AskH3w1 (when the above three events did not happen), it means that only executions with an instance of $\mathcal{S}$ may lead to acceptance (and either $\mathcal{C}$ or the adversary). Exactly the same analysis as for AskH1-Passive and AskH1-WithS leads to

$$\Pr[\mathsf{AskH3w1}_4] \leq \frac{q_p + q_{\mathcal{S}}}{2^l} \ . \tag{26}$$

About AskH4w13 (when both of the events AskH1 and AskH3w1 did not happen), the same analysis as for AskH3w1 leads to

$$\Pr[\mathsf{AskH4w13}_4] \leq \frac{q_p + q_{\mathcal{S}}}{2^l} \ . \tag{27}$$

As a conclusion, we get an upper-bound for the probability of AskH_4 by combining all the cases:

$$\Pr[\mathsf{AskH}_4] \leq \frac{3q_p + 3q_{\mathcal{S}} + q_{\mathcal{C}}}{2^l} \ . \tag{28}$$

Combining equations (6), (7), (22) and (28), one gets either

$$\Pr[\mathsf{A}_0] \leq \frac{q_{\mathcal{C}}}{2^{k_1}} + \Delta \qquad \Pr[\mathsf{SwA}_0] = \frac{1}{2} + \Delta \ , \tag{29}$$

where

$$\Delta \leq \frac{3q_p + 3q_{\mathcal{S}} + q_{\mathcal{C}}}{2^l} + \frac{q_h^2 + (q_{\mathcal{C}} + q_p)^2 + q_g'^2}{2^{l+1}} \ . \tag{30}$$

One can get the result as desired by noting that $\Pr[\mathsf{S}_0] \leq \Pr[\mathsf{SwA}_0] + \Pr[\mathsf{A}_0]$.

It is of practical significance to note that in the real world applications the Leak-query is limited by the *physical* power of an adversary which are usually much less, whereas the Send and Execute-queries are directly relevant to interactions with the parties.

References

1. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure against Dictionary Attacks. In *Proc. of EUROCRYPT 2000*, LNCS 1807, pages 139-155. Springer-Verlag, 2000.
2. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *Proc. of ACM CCS '93*, pages 62-73, 1993.
3. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *Proc. of CRYPTO '93*, LNCS 773, pages 232-249. Springer-Verlag, 1993.
4. R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. In *Proc. of the 30th ACM Symposium on Theory of Computing (STOC)*, pages 209-218, ACM, 1998.
5. D. Catalano, D. Pointcheval, and T. Pornin. Trapdoor Hard-to-Invert Group Isomorphisms and Their Application to Password-based Authentication. *Journal of Cryptology*, 2006. The extended abstract appeared at CRYPTO 2004. Available at `http://www.di.ens.fr/~pointche/pub.php?reference=CaPoPo06`.
6. S. Halevi and H. Krawczyk. Public-Key Cryptography and Password Protocols. *ACM Transactions on Information and System Security*, Vol. 2, No. 3, pages 230-268. ACM Press, August 1999.
7. A. Shamir. How to Share a Secret. In *Proc. of Communications of the ACM*, Vol. 22(11), pages 612-613, 1979.
8. V. Shoup. OAEP Reconsidered. *Journal of Cryptology*, Vol. 15(4), pages 223-249, September 2002.
9. S. Shin, K. Kobara, and H. Imai. Efficient and Leakage-Resilient Authenticated Key Transport Protocol Based on RSA. In *Proc. of ACNS2005*, LNCS 3531, pages 269-284. Springer-Verlag, 2005.
10. Q. Tang and Chris J. Mitchell. Weaknesses in a Leakage-Resilient Authenticated Key Transport Protocol. `http://eprint.iacr.org/2005/173`.