Chapter 9
Parallel Signcryption

Josef Pieprzyk and David Pointcheval

9.1 Introduction

The primary motivation for signcryption was the gain in efficiency when both encryption and signing need to be performed. These two cryptographic operations may be done sequentially either by first encrypt and then sign ($\mathcal{E}t\mathcal{S}$) or alternatively, by first sign and then encrypt ($\mathcal{S}t\mathcal{E}$). Further gains in efficiency can be achieved if encryption and signature are carried out in parallel ($\mathcal{E}\&\mathcal{S}$). More importantly, however, is that these efficiency gains are complemented by gains in security, i.e., we may use relative weak encryption and signature schemes in order to obtain a "stronger" signcryption scheme. The reader is referred to Chaps. 2 and 3 for a discussion of the different "strengths" of security model (e.g., outsider vs. insider adversaries, two-user vs. multi-user setting).

9.2 Concept of Parallel Signcryption

Efficiency and security are the two main requirements for cryptographic algorithms. Striking the balance between the two requirements is the real challenge. New ever-growing Internet applications such as distance learning, video streaming, e-commerce, e-government, e-health, etc., heavily rely on sophisticated protocols whose explicit goals are the fast, reliable, and secure delivery of large volumes of data.

Cryptographic protocols can be sped up by

- designing new, faster secure cryptographic algorithms—this option is not always available as once an algorithm becomes a standard or has been incorporated into the protocol: the designers are stuck with it for some time,

J. Pieprzyk (✉)
Department of Computing, Center for Advanced Computing – Algorithms and Cryptography, Macquerie University, Sydney, Australia
e-mail: josef@comp.mq.edu.au

- parallelizing operations required by the cryptographic algorithms—this approach can be applied at the level of a single algorithm (parallel thread implementation) and/or at the level of the protocol (parallel execution of the protocol components).

Privacy and authenticity are two basic security goals. As already discussed in the motivation for signcryption, there are many applications that require both goals to be achieved simultaneously. However, the main problem considered initially was the design of encryption and signature so that their concatenation maximizes savings of computing resources. Our goal here is to achieve the lower bound in terms of time necessary to perform authenticated encryption and decryption, or

$$\begin{gathered}\text{time(parallel } \texttt{Encrypt\&}\ \texttt{Sign}) \approx \max\{\text{time}(\texttt{Encrypt}), \text{time}(\texttt{Sign})\} \\ \text{and} \\ \text{time(parallel } \texttt{Decrypt\&}\ \texttt{Verify}) \approx \max\{\text{time}(\texttt{Decrypt}), \text{time}(\texttt{Verify})\}\end{gathered}$$

At best, one would expect that parallel encryption and signing will consume roughly the same time as the most time-consuming operation (either signing or encryption for the signcryption operation and either verifying or decrypting for the unsigncryption operation).

The parallel encryption and signing methodology was introduced by An et al. [10]—see Chap. 2 for a detailed discussion of their results. Independently, the concept was also developed by Pieprzyk and Pointcheval [160]. Both works can be seen as generalizations of the signcryption concept introduced by Zheng [203, 204]. An et al. [10] developed a security model for parallel signcryption and present the commit-then-encrypt-and-sign ($\mathcal{C}t\mathcal{E}\&\mathcal{S}$) scheme that uses three cryptographic blocks: a commitment scheme, a public key encryption scheme, and a signature scheme (as described in Chap. 6). The solution given by Pieprzyk and Pointcheval [160] implements the commitment part very efficiently using secret sharing. It also shows how to combine encryption and signing so that they strengthen each other and can be executed in parallel.

9.3 Overview of Constructions

A trivial implementation of parallel signcryption could be as simple as applying encrypt and sign operations to the same message in parallel. This, of course, does not work as the signature may reveal the message (see Chap. 2).

A classical solution could be the well-known *envelope technique* (see Fig. 9.1) that first defines a secret session key. This key is encrypted under the public key and is used, in parallel, to encrypt, under a symmetric encryption, a message and a signature on it. If one assumes that the symmetric encryption has a negligible cost (some may disagree with this claim), then this allows parallel encryption and signing. For unsigncryption, the recipient first decrypts the session key and then extracts the message and the signature. Only when all that operations have been completed,

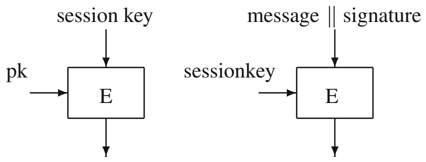

Fig. 9.1 Envelope technique

can one verify the signature. Therefore, decryption and verification cannot be done in parallel.

The commit-then-encrypt-and-sign ($\mathcal{C}t\mathcal{E}\&\mathcal{S}$) [10] is a little bit better. It is shown in Fig. 9.2. The signcryption algorithm first commits to the message m, computing the actual committed value c and the decommitment d (see Sect. 6.4.2). It then encrypts d in e and signs c getting s. The unsigncryption algorithm can unsigncrypt the ciphertext (e, c, s) by first verifying (c, s) and decrypting e into d. The decommitment d finally helps to recover m (by opening c). However, the opening algorithm may not be as efficient as required.

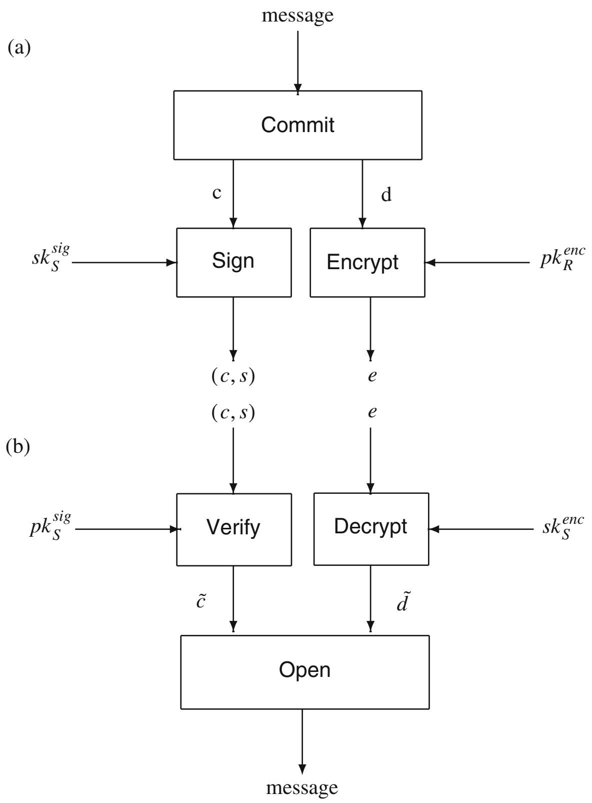

Fig. 9.2 The commit-then-encrypt-and-sign signcryption **a** encrypt and sign, **b** decrypt and verify

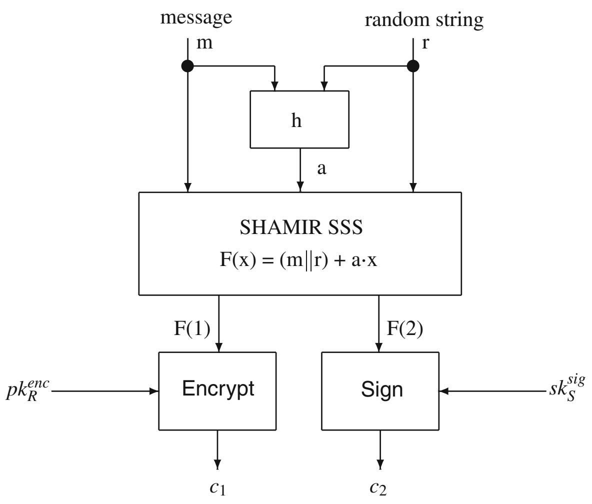

Fig. 9.3 Generic signcryption

The two constructions presented in this chapter are in the same vein as those presented in Chap. 6. They apply an efficient commitment scheme (proven secure in the random oracle model [29]) which allows for weak assumptions about the underlying encryption and signature schemes. The commitment scheme is based on a (2, 2) Shamir secret sharing scheme (see Fig. 9.3). In a (k, n) Shamir secret sharing scheme, a secret is shared among n parties. Any k parties out of n can recover the secret while any group of less than k parties has no information about the secret. The (k, n) Shamir secret sharing scheme [176] simply exploits Lagrange interpolation of polynomials of degree $k - 1$.

As we use a (2, 2) Shamir secret sharing, we need a linear polynomial whose coefficients are strongly related to the message m in a randomized way. For a random string r, the constant coefficient is $(m\|r)$ and the linear coefficient is $h(m\|r)$, where h is a hash function returning values from $\mathbb{Z}_p$. The polynomial, over $\mathbb{Z}_p$, evaluated at two points produces two shares. One of the shares is encrypted and the other is authenticated (in parallel). The perfectness of Shamir secret sharing guarantees that knowledge of one of the shares provides no information (in the information-theoretic sense) about the constant coefficient (the secret), and consequently no information about the message m.

9.4 Generic Parallel Signcryption

9.4.1 Description of the Scheme

The signcryption scheme uses the following building blocks:

- an encryption scheme $\mathcal{E} = (\texttt{EncKeyGen}, \texttt{Encrypt}, \texttt{Decrypt})$,
- a signature scheme $\mathcal{S} = (\texttt{SigKeyGen}, \texttt{Sign}, \texttt{Verify})$,
- a large $(k + 1)$-bit prime p, which defines the field $\mathbb{Z}_p$ with $p \geq 2^k$,

- a hash function $h : \mathbb{Z}_p \to \mathbb{Z}_p$,
- two integers k_1 and k_2 that are two security parameters such that $k = k_1 + k_2$.

We will use a signature scheme with message recovery (see Sect. 1.3.2). This means that the verification algorithm `Verify` takes as input a signature s and a public key pk^{sig}. It outputs either a message m indicating that the signature is valid for message m or the error symbol $\perp$.

The signcryption scheme is defined by the following collection of algorithms:

- $\mathtt{KeyGen}(1^k)$: Compute $(sk^{sig}, pk^{sig}) \overset{R}{\leftarrow} \mathtt{KeyGen}_S(1^k) \overset{\text{def}}{=} \mathtt{SigKeyGen}(1^k)$ and $(sk^{enc}, pk^{enc}) \overset{R}{\leftarrow} \mathtt{KeyGen}_R(1^k) \overset{\text{def}}{=} \mathtt{EncKeyGen}(1^k)$. The sender keys are

$$(sk_S, pk_S) \overset{\text{def}}{=} (sk^{sig}, pk^{sig})$$

  and the receiver keys are

$$(sk_R, pk_R) \overset{\text{def}}{=} (sk^{enc}, pk^{enc})$$

  We now consider two users, the sender with keys (sk_S, pk_S) and the receiver with keys (sk_R, pk_R).
- $\mathtt{Signcrypt}(sk_S, pk_R, m)$: Given a message $m \in \{0,1\}^{k_1}$ that needs to be encrypted and signed:

  1. Choose a random integer $r \in \{0,1\}^{k_2}$ and compute $a = h(m\|r) \in \mathbb{Z}_p$, where $(m\|r) \in \{0,1\}^k \subseteq \mathbb{Z}_p$.
  2. Form an instance of a $(2, 2)$ Shamir secret sharing scheme over $\mathbb{Z}_p$ with the polynomial $F(x) = (m\|r) + ax \bmod p$. Define two shares $s_1 \leftarrow F(1)$ and $s_2 \leftarrow F(2)$ in $\mathbb{Z}_p$.
  3. Calculate in parallel $c_1 \leftarrow \mathtt{Encrypt}(pk_R, s_1)$ and $c_2 \leftarrow \mathtt{Sign}(sk_S, s_2)$. The ciphertext (c_1, c_2) is dispatched to the receiver R.

- $\mathtt{Unsigncrypt}(pk_S, sk_R, (c_1, c_2))$:

  1. Perform decryption and signature verification in parallel:

$$t_1 \leftarrow \mathtt{Decrypt}(sk_R, c_1)$$

  and

$$t_2 \leftarrow \mathtt{Verify}(pk_S, c_2)$$

  Note that both the `Decrypt` and `Verify` algorithms return integers in $\mathbb{Z}_p$ unless a failure occurs. Indeed, it is possible that `Decrypt` returns $\perp$ if it decides that the ciphertext is invalid. Similarly, `Verify` returns a message (as the signature scheme is assumed to have message recovery) or $\perp$ if the signature is invalid. In case of at least one failure, the decryption and verifying algorithm `Unsigncrypt` returns $\perp$ and stops.

2. Given the two points $(1, t_1)$ and $(2, t_2)$, use the Lagrange interpolation and find the polynomial $\tilde{F}(x) = a_0 + a_1 x \bmod p$ for which these two points are solutions (i.e., compute $a_0 = 2t_1 - t_2$ and $a_1 = t_2 - t_1$).
3. Check whether $a_1 = h(a_0)$ or equivalently if $t_2 - t_1 = h(2t_1 - t_2)$. If the check holds, extract m from a_0 (as $a_0 = (m \| r)$) and return m. Otherwise, output $\perp$.

9.4.2 Security Analysis

The original research of Pieprzyk and Pointcheval [160] proved the following theorem:

Theorem 9.1 *If the encryption scheme is* IND-CCA2 *and the signature scheme is* sUF-RMA*, then the generic parallel signcryption scheme is* IND-CCA *and* UF-CMA *secure in the outsider security model for the two-user setting.*

The security model in this theorem is unfortunately weak, i.e., outsider security for the two-user setting, without FSO/FUO. However, the multi-user setting with FSO/FUO is covered if both ID_S and ID_R are included in the hash value, i.e., $a = h(ID_S \| ID_R \| m \| r)$. Furthermore, in the case of a deterministic signature, one even gets insider security:

Theorem 9.2 *If the encryption scheme is* IND-CCA2 *and the signature scheme is deterministic and* UF-RMA*, then the generic parallel signcryption scheme, as modified above, is* FSO/FUO-IND-CCA2 *and* FSO/FUO-UF-CMA *secure in the insider security model for the multi-user setting.*

More precisely, we are going to prove the two following results.

Lemma 9.1 *Suppose there exists an insider adversary $\mathcal{A}$ against* FSO/FUO-UF-CMA *security of the generic parallel signcryption scheme, in the multi-user setting, with advantage $Adv_{Signcrypt}^{UF-CMA}(k)$ whose running time is bounded by t and asks at most q_h queries to the random oracle h and q_{sc} queries to the signcryption oracle. Then, there exists an adversary $\mathcal{B}$ against the* UF-RMA *security of the signature scheme with advantage $Succ_{Sign}^{UF-RMA}(k)$ whose running time is bounded by $t' \le t + q_{sc}(\tau + O(1))$, where τ denotes the maximal running time of the encryption and signing algorithms, and that asks at most q_{sc} queries to its signature oracle, for which*

$$Adv_{Signcrypt}^{UF-CMA}(k) \le Succ_{Sign}^{UF-RMA}(k) + 2q_{sc} \times \frac{q_h + q_{sc}}{2^{k_2}}.$$

Lemma 9.2 *Suppose there exists an insider adversary $\mathcal{A}$ against* FSO/FUO-IND-CCA2 *of the generic parallel signcryption scheme, in the multi-user setting, with* *advantage $Adv_{Signcrypt}^{IND-CCA2}(k)$ whose running time is bounded by t and who asks at most q_h queries to the random oracle h and q_{usc} queries to the unsigncryption oracle. Then there exists an attacker $\mathcal{B}$ against the* IND-CCA2 *security of the encryption scheme with advantage $Adv_{Encrypt}^{IND-CCA2}(k)$ whose running time is bounded by t' and that makes at most q_{usc} queries to the unsigncryption oracle, where $t' \leq t + q_{usc}(\tau + O(1))$ and τ denotes the maximum running time of the decryption and verification algorithms, and*

$$Adv_{Signcrypt}^{IND-CCA2}(k) \leq 2 \times Adv_{Encrypt}^{IND-CCA2}(k) + \frac{q_h + q_{usc}}{2^{k_2-1}}$$

if the signature scheme is deterministic.

We prove the above lemmas in the random oracle model. When a random oracle h is called, we have two possibilities. One possibility is that the query has been already asked. In this case the answer has already been defined by the simulation and the same answer has to be given. The second possibility is that the query has not been asked. In this case, a random value in $\mathbb{Z}_p$ is given. Of course, one has to be careful when defining an answer of a random oracle as the following conditions have to be satisfied:

- this answer must not have already been defined and
- the answer must be uniformly distributed.

Furthermore, we denote by q_H the number of answers defined for h. This number will be easily upper bounded by $q_h + q_{sc} + q_{usc}$ in the following simulations.

Proof (of Lemma 9.1) Assume that after q_{sc} queries to the oracle `Signcrypt`, an adversary $\mathcal{A}$ outputs a new ciphertext (c_1, c_2), which is valid with probability $Adv_{Signcrypt}^{UF-CMA}(k)$. We use the adversary to perform an existential forgery (under a random message attack) against the signature scheme $\mathcal{S}$. For this proof, we consider the multi-user insider security model. Hence, the attacker knows the public key pk_S of the target sender ID_S^* and has access to the signcryption oracle under sk_S.

We first design a simulator $\mathcal{B}$ which has access to a list of message–signature pairs, produced by the signing oracle under sk_S (the messages are assumed to have been randomly drawn from $\mathbb{Z}_p$ and not chosen by the adversary). It is given the private/public keys (sk_R, pk_R) produced by the adversary, for the encryption scheme, and is also provided with the public key pk_S of the signature scheme. Any query m by $\mathcal{A}$ to the oracle `Signcrypt` under sk_S, for any recipient ID_R, can be simulated using a new valid message–signature pair (M, S), for the signature scheme. Indeed, M is defined to be s_2 and S is defined to be c_2. Then, one chooses a random r. Since

$$s_2 = (m\|r) + 2h(ID_S^*\|ID_R\|m\|r) \bmod p = M$$

one needs to define the random oracle at the point $(ID_S^*\|ID_R\|m\|r)$ (unless it has already been done, which then raises the event BADH). So we get

$$h(ID_S^*\|ID_R\|m\|r) \leftarrow \frac{M-(m\|r)}{2} \bmod p$$

and therefore

$$s_1 \leftarrow (m\|r) + h(ID_S^*\|ID_R\|m\|r) = \frac{M+(m\|r)}{2} \bmod p$$

Using the public key of the encryption scheme for the recipient ID_R, one can encrypt s_1 to obtain c_1. The pair (c_1, c_2) is a valid ciphertext of m.

Finally, the adversary $\mathcal{A}$ returns a ciphertext (c_1, c_2) for a new message m', for ID_R^* from ID_S^*, which is valid with probability $Adv_{Signcrypt}^{UF-CMA}(k)$. With the public key of the signature scheme, one can extract the message s_2 from c_2. By definition, (s_2, c_2) is an existential forgery for the signature scheme. Indeed, one just has to check whether it is a really new signed message. If this is not a new signed message, then s_2 has already been signed by the oracle `Signcrypt` for $m\|r$, where

$$s_2 = (m\|r) + 2h(ID_S^*\|ID_R\|m\|r) \bmod p$$

Note that s_2 is uniquely defined in the list of the queries asked to the random oracle h unless one has found a collision for the function

$$G : (x, y) \mapsto x + 2h(ID_S^*\|y\|x) \bmod p$$

among the q_{sc} values given by the simulation and the q_H answers obtained by the adversary (either directly from queries or implicitly defined by the simulation). Because of the randomness of the random oracle h, this is upper bounded by $q_{sc} \cdot q_H/2^k$.

Furthermore, one has to be sure that everything looks like in a real attack from the adversary $\mathcal{A}$ point of view. However, when one defines a value for h, it may have already been defined (event BADH). Because of the randomness of r, the probability of such an event is less than $q_H/2^{k_2}$ for each simulation of the oracle `Signcrypt`.

Finally, the probability for $\mathcal{B}$ to produce an existential forgery against the signature scheme is greater than

$$\begin{aligned} Adv_{Sign}^{UF-RMA}(k) &\geq Adv_{Signcrypt}^{UF-CMA}(k) - q_{sc} \cdot q_H \times \left(\frac{1}{2^{k_2}} + \frac{1}{2^k}\right) \\ &\geq Adv_{Signcrypt}^{UF-CMA}(k) - 2q_{sc} \cdot q_H \times \left(\frac{1}{2^{k_2}}\right) \end{aligned}$$

Furthermore, one can easily see that $q_H \leq q_h + q_{sc}$, hence the result. □

Proof (of Lemma 9.2) Assume that an adversary $\mathcal{A}$ has made q_{usc} queries to the oracle `Unsigncrypt`. $\mathcal{A}$ also has chosen a pair of messages m_0 and m_1 and has received a ciphertext (c_1^*, c_2^*) under (sk_S, pk_R) of either m_0 or m_1. The unknown message is denoted by m_b, where b is the bit the adversary wishes to find out. The adversary outputs a bit d which is equal to b with advantage ε such that $\Pr[d = b] = 1/2 + \varepsilon$.

We work with the multi-user insider security model. The attacker receives a target receiver ID_R^* public key pk_R and has access to the unsigncryption oracle under sk_R.

We design a simulator $\mathcal{B}$ which is given the public key pk_R of the encryption scheme and has access to the decryption oracle `Decrypt` (under sk_R).

Any call by $\mathcal{A}$ to the oracle `Unsigncrypt` under sk_R, from any sender ID_S, can be simulated using the decryption oracle `Decrypt` access. Indeed, for a query (c_1, c_2), one first asks the query c_1 to the oracle `Decrypt` and obtains s_1. Thanks to the public key of the signature scheme, one can get s_2 from c_2. This is enough to check the validity of the ciphertext (c_1, c_2) and to decrypt it. We will see later if this simulation is always possible.

Let us first show how one generates the challenge ciphertext. When $\mathcal{B}$ receives the pair of messages m_0 and m_1 from $\mathcal{A}$, it randomly chooses two random integers r_0 and r_1 to produce two new messages for the encryption scheme, namely

$$\begin{aligned} M_0 &\leftarrow (m_0\|r_0) + h(ID_S^*\|ID_R^*\|m_0\|r_0) \bmod p \\ M_1 &\leftarrow (m_1\|r_1) + h(ID_S^*\|ID_R^*\|m_1\|r_1) \bmod p \end{aligned}$$

$\mathcal{B}$ receives the ciphertext c_1^* of M_b and has to guess the bit b, with the help of $\mathcal{A}$. For that, it chooses a random bit b' (hoping it to be equal to b) and defines

$$s_2^* \leftarrow (m_{b'}\|r_{b'}) + 2h(ID_S^*\|ID_R^*\|m_{b'}\|r_{b'}) \bmod p$$

Then, it signs it using the private key sk_S of the signature scheme and gets c_2^*. Next it sends the pair (c_1^*, c_2^*) as a ciphertext of m_b (for the unknown bit b). Finally, the adversary $\mathcal{A}$ ends its attack, returning a bit d to $\mathcal{B}$ and $\mathcal{B}$ forwards it as its final answer.

The simulation of $\mathcal{A}$'s unsigncryption queries by $\mathcal{B}$ works fine for any query (c_1, c_2) with $c_1^* \neq c_1$, as shown above. The above simulation breaks for queries (c_1^*, c_2), as the decryption oracle is prevented to be queried for the challenge ciphertext c_1^* while the oracle `Unsigncrypt` accepts queries as long as $c_1 \neq c_1^*$, or $c_2 \neq c_2^*$, or $ID_S \neq ID_S^*$. If $\mathcal{A}$ submits an unsigncryption oracle query of them (c_1^*, c_2) then the simulator $\mathcal{B}$ returns $\perp$. The event that $\mathcal{B}$ rejects an unsigncryption oracle (c_1^*, C_2) which is actually valid is called BADD. Later, we will show that this happens with a negligible probability.

Now, we study the advantage of the simulator $\mathcal{B}$ in breaking IND-CCA2 of the encryption scheme, which is

$$\begin{aligned} Adv_{Encrypt}^{IND-CCA2}(k) &= \Pr[d = b] - \tfrac{1}{2} \\ &\geq \Pr[d = b \wedge \neg\text{BADD}] - \tfrac{1}{2} \\ &\geq \Pr[d = b \,|\, \neg\text{BADD}] - \Pr[\text{BADD}] - \tfrac{1}{2} \\ &= \tfrac{1}{2} \cdot \Pr[d = b \,|\, b = b' \wedge \neg\text{BADD}] + \tfrac{1}{2} \cdot \Pr[d = b \,|\, b \neq b' \wedge \neg\text{BADD}] \\ &\qquad - \Pr[\text{BADD}] - \tfrac{1}{2} \end{aligned}$$

Let us now examine each term. First note that, when $b' = b$, the simulated challenge (c_1^*, c_2^*) is identical to a real challenge:

$$\begin{aligned}\varepsilon &= \Pr[d = b \mid b' = b] - 1/2\\ &\leq \Pr[d = b \mid b' = b \wedge \neg\textsc{BadD}] + \Pr[\textsc{BadD}] - 1/2\end{aligned}$$

Let us now focus on the second term in the inequality (when $b' \neq b$), by defining $\textsc{AskH}$ to be the event that the adversary $\mathcal{A}$ either asks $(ID_S^* \| ID_R^* \| m_0 \| r_0)$ or $(ID_S^* \| ID_R^* \| m_1 \| r_1)$ to the random oracle h. It is equal to

$$\begin{aligned}&\Pr[d = b \mid b' \neq b \wedge \neg\textsc{BadD}]\\ &\quad \geq \Pr[d = b \mid b' \neq b \wedge \neg\textsc{BadD} \wedge \neg\textsc{AskH}]\\ &\qquad\qquad \times \Pr[\neg\textsc{AskH} | b' \neq b \wedge \neg\textsc{BadD}]\end{aligned}$$

Clearly, in the case that $b' \neq b$, the adversary may have some information (in the theoretical sense) about

$$\begin{aligned}M_b &= (m_b \| r_b) + h(ID_S^* \| ID_R^* \| m_b \| r_b) \bmod p\\ s_2^* &= (m_{b'} \| r_{b'}) + 2h(ID_S^* \| ID_R^* \| m_{b'} \| r_{b'}) \bmod p\end{aligned}$$

However, without the event $\textsc{AskH}$, the hash values perfectly hide the first part and therefore the answer of $\mathcal{A}$ is independent of b (a random variable):

$$\Pr[d = b \mid b' \neq b \wedge \neg\textsc{BadD} \wedge \neg\textsc{AskH}] = \frac{1}{2}$$

On the other hand, as we have said above, the value $h(ID_S^* \| ID_R^* \| m_i \| r_i)$ perfectly hides $(m_i \| r_i)$, for $i = 0, 1$, and therefore one cannot get any information about the random values r_0 and r_1 without a guess. The event $\textsc{AskH}$ happens with the probability less than $2q_H/2^{k_2}$. We therefore conclude

$$\Pr[\neg\textsc{AskH} | b' \neq b \wedge \neg\textsc{BadD}] \geq 1 - 2q_H/2^{k_2}$$

Finally, let us examine the probability $\Pr[\textsc{BadD}]$ of a wrong decryption reject: $c_1 = c_1^*$ but $c_2 \neq c_2^*$ or $ID_S \neq ID_S^*$. Since this should be a valid signature, c_2 is the signature of some element s_2 and c_1 is the encryption of M_b such that $s_2 - M_b = h(ID_S \| ID_R^* \| 2M_b - s_2)$.

Because of the random oracle h, the probability to find such a pair (ID_S, s_2) is less than $q_H/2^k$, except the constructed pair (ID_S^*, s_2^*). But since the signature scheme is deterministic, then $c_2 = c_2^*$ and $ID_S = ID_S^*$, and such a query cannot be asked to the unsigncryption oracle. As a consequence,

$$\Pr[\textsc{BadD}] \leq \frac{q_H}{2^k}$$

Now we collect all the terms and get

$$\begin{aligned}Adv_{Encrypt}^{IND-CCA2}(k) &\geq \frac{1}{2}\left(\varepsilon - \Pr[\textsc{BadD}] + \frac{1}{2}\right)\\ &\quad + \frac{1}{2}\left(\frac{1}{2}\Pr[\neg\textsc{AskH}|b' \neq b \wedge \neg\textsc{BadD}]\right) - \Pr[\textsc{BadD}] - \frac{1}{2}\\ &\geq \frac{\varepsilon}{2} - \frac{3}{2}\Pr[\textsc{BadD}] - \frac{q_H}{2^{k_2+1}}\\ &\geq \frac{1}{2}Adv_{Signcrypt}^{IND-CCA2}(k) - \frac{3 \cdot q_H}{2 \cdot 2^k} - \frac{q_H}{2 \cdot 2^{k_2}}\end{aligned}$$

This concludes the proof, since $q_H \leq q_h + q_{usc}$. □

From the efficiency point of view, this generic scheme is almost optimal since on the sender side, only one hash value and two additions are required before the parallel encryption and signature processes. The process needed on the receiver side reaches the same kind of optimality. However, the security requirements of the basic schemes, the encryption scheme $\mathcal{E}$ and the signature scheme $\mathcal{S}$, are very strong. Indeed, the encryption scheme is required to be semantically secure against chosen-ciphertext attack and the signature scheme must already prevent existential forgeries.

9.5 Optimal Parallel Signcryption

Adding a kind of OAEP technique [30], we can improve the generic scheme, in the sense that we can weaken the security requirements of the basic primitives. The new proposal just requires the encryption scheme to be deterministic and one way against chosen-plaintext attack, which is a very weak security requirement—even the plain RSA encryption scheme [165] achieves it under the RSA assumption. The signature scheme is required to prevent universal forgeries under the random message attack—the plain RSA signature scheme achieves this security level.

9.5.1 Description of the Scheme

The scheme is illustrated in Fig. 9.4. The building blocks are

- an encryption scheme $\mathcal{E} = (\texttt{EncKeyGen}, \texttt{Encrypt}, \texttt{Decrypt})$,
- a signature scheme $\mathcal{S} = (\texttt{SigKeyGen}, \texttt{Sign}, \texttt{Verify})$,
- a large k-bit prime p, which defines the field $\mathbb{Z}_p$, with $p \geq 2^k$,
- two integers k_1 and k_2 that are security parameters such that $k = k_1 + k_2$,
- hash functions

$$f : \{0,1\}^k \to \{0,1\}^k,\ g : \{0,1\}^k \to \{0,1\}^k \text{ and } h : \{0,1\}^* \to \{0,1\}^{k_2}$$

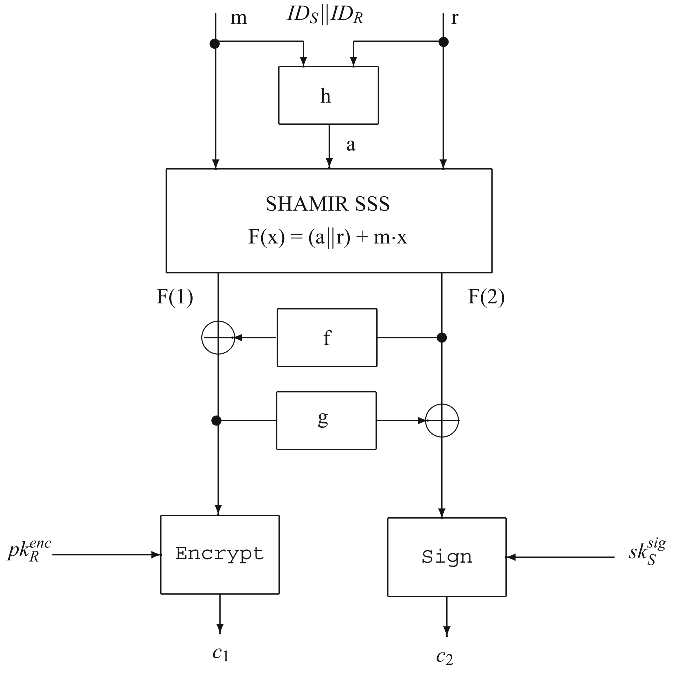

Fig. 9.4 Optimal signcryption (encryption and signing)

The signcryption scheme works as follows:

- `KeyGen`(1^k) : Compute $(sk^{sig}, pk^{sig}) \overset{R}{\leftarrow}$ `KeyGen`$_S(1^k) \overset{\mathsf{def}}{=}$ `SigKeyGen`(1^k) and $(sk^{enc}, pk^{enc}) \overset{R}{\leftarrow}$ `KeyGen`$_R(1^k) \overset{\mathsf{def}}{=}$ `EncKeyGen`(1^k). The sender keys are

$$(sk_S, pk_S) \overset{\mathsf{def}}{=} (sk^{sig}, pk^{sig})$$

  and the receiver keys are

$$(sk_R, pk_R) \overset{\mathsf{def}}{=} (sk^{enc}, pk^{enc})$$

  We now consider two users, the sender with the keys (sk_S, pk_S) and the receiver with the keys (sk_R, pk_R).
- `Signcrypt`(sk_S, pk_R, m): Given a message $m \in \mathbb{Z}_p$ that needs to be encrypted and signed:

  1. Choose a random integer $r \in \{0, 1\}^{k_1}$ and compute $a = h(ID_S \| ID_R \| m \| r)$.
  2. Form an instance of a (2, 2) Shamir secret sharing scheme over $\mathbb{Z}_p$ with the polynomial $F(x) = (a \| r) + mx \bmod p$. Define two shares $s_1 \leftarrow F(1)$ and $s_2 \leftarrow F(2)$.

3. Compute the transform $r_1 \leftarrow s_1 \oplus f(s_2)$ and $r_2 \leftarrow s_2 \oplus g(r_1)$.
4. Calculate (in parallel) $c_1 \leftarrow \texttt{Encrypt}(pk_R, r_1)$ and $c_2 \leftarrow \texttt{Sign}(sk_S, r_2)$. The ciphertext (c_1, c_2) is then dispatched to the receiver R.

- $\texttt{Unsigncrypt}(pk_S, sk_R, (c_1, c_2))$:

1. Perform decryption and signature verification in parallel so

$$u_1 \leftarrow \texttt{Decrypt}(sk_R, c_1)$$

and

$$u_2 \leftarrow \texttt{Verify}(pk_S, c_2)$$

Note that both the `Decrypt` and `Verify` algorithms return integers in $\mathbb{Z}_p$ unless some failure occurs. It is possible that `Decrypt` returns $\perp$ if it decides that the ciphertext is invalid. Similarly, `Verify` returns a message (as we are using a signature with message recovery) or $\perp$ if the signature is invalid. In the case of a failure, the `Unsigncrypt` algorithm returns $\perp$ and stops.
2. Compute the inversion $t_2 \leftarrow u_2 \oplus g(u_1)$ and $t_1 \leftarrow u_1 \oplus f(t_2)$.
3. Knowing two points $(1, t_1)$ and $(2, t_2)$, use the Lagrange interpolation and find the polynomial $\tilde{F}(x) = a_0 + a_1 x \bmod p$, where $a_0 = 2t_1 - t_2$ and $a_1 = t_2 - t_1$.
4. Extract r from a_0 and check whether $h(ID_S \| ID_R \| a_1 \| r) \| r = a_0 \bmod p$. If the check holds, return a_1 as m. Otherwise, return $\perp$.

9.5.2 Security Analysis

The following theorem characterizes the security of the optimal parallel signcryption. (Recall that the universal forgery notion for a signature scheme is discussed in Sect. 1.3.2.)

Theorem 9.3 *If the encryption scheme is deterministic and* OW-CPA *secure and the signature scheme is deterministic and* uUF-RMA *secure, then the optimal parallel signcryption scheme is* FSO/FUO-IND-CCA2 *and* FSO/FUO-UF-CMA *secure in the insider security model for the multi-user setting.*

More precisely, one can prove the two following results.

Lemma 9.3 *Consider an insider adversary $\mathcal{A}$ against the* FSO/FUO-UF-CMA *security of the optimal parallel signcryption scheme, in the multi-user setting, with advantage $Adv^{UF-CMA}_{Signcrypt}(k)$ whose running time is bounded by t and who makes at most q_h queries to the random oracle h, q_g queries to the random oracle g, and q_{sc} queries to the signcryption oracle. Then there exists an attacker $\mathcal{B}$ against the* uUF-RMA *security of the signature scheme with advantage $Adv^{uUF-RMA}_{Sign}(k)$ whose running time is bounded by $t' \leq t + q_{sc}(\tau + O(1))$, where τ denotes the* *maximal running time of the encryption and signing algorithms, and that makes at most $q_h + q_{sc}$ queries to the signing oracle, for which*

$$Adv_{Signcrypt}^{UF-CMA}(k) \leq (q_h + q_{sc}) \times Adv_{Sign}^{uUF-RMA}(k) + \frac{(q_g + q_h + q_{sc})^2 + 2}{2^{k_2}}$$

Lemma 9.4 *Consider an insider adversary $\mathcal{A}$ against the* FSO/FUO-IND-CCA2 *security of the generic parallel signcryption scheme, in the multi-user setting, with advantage $Adv_{Signcrypt}^{IND-CCA2}(k)$ whose running time is bounded by t and which makes at most q_h queries to the random oracle h and q_{usc} queries to the unsigncryption oracle. Then there exists an attacker $\mathcal{B}$ against the* OW-CPA *security of the public key encryption scheme with advantage $Adv_{Encrypt}^{OW-CPA}(k)$ and whose running time is bounded by $t' \leq t + q_{usc}(\tau + O(1))$, where τ denotes the maximal running time of the decryption and verification algorithms, for which*

$$Adv_{Signcrypt}^{IND-CCA2}(k) \leq Adv_{Encrypt}^{OW-CPA}(k) + \frac{q_h}{2^{k_1}} + \frac{q_{usc}}{2^{k_2}}$$

The proofs are similar to the proofs of Lemmas 9.1 and 9.2. Again, we are in the random oracle model, and the functions f, g, and h are modeled by random oracles. The number of queries to these oracles is q_f, q_g, and q_h, respectively. Furthermore, we denote by q_F, q_G, and q_H the number of answers defined for f, g, and h, respectively.

Proof (of Lemma 9.3) Assume that after q_{sc} queries to oracle `Signcrypt`, an adversary $\mathcal{A}$ outputs a new ciphertext (c_1, c_2), which is valid with the probability $Adv_{Signcrypt}^{UF-CMA}(k)$. We use this adversary to perform a universal forgery that produces a new signature on a designated random message μ (under a known random message attack) against the signature scheme $\mathcal{S}$. Since we are dealing with the insider security model, the adversary has a target sender ID_S^* in mind and he/she knows the sender public key pk_S. The adversary has access to the signcryption oracle under sk_S. Now we design a simulator $\mathcal{B}$, which has access to a list of message–signature pairs, produced by the signing oracle (the messages are assumed to have been randomly drawn from $\mathbb{Z}_p$ and not chosen by the adversary). It is given the private/public keys (sk_R, pk_R) produced by the adversary, for the encryption scheme of the target receiver ID_R^*. Note that a valid ciphertext must satisfy the equality $h(ID_S \| ID_R \| m \| r) \| r = a_0 \bmod p$. Therefore, the probability of getting a valid ciphertext (from ID_S^* to ID_R^*) without asking $h(ID_S^* \| ID_R^* \| m \| r)$ is at most 2^{-k_2}. The query $(ID_S^* \| ID_R^* \| m \| r)$ must have been asked to the oracle h with a probability greater than $Adv_{Signcrypt}^{UF-CMA}(k) - 2^{-k_2}$. It is provided with the public key pk_S of ID_S^* for the signature scheme. It is furthermore given a list of q_H message–signature (M, S) pairs, where messages are randomly chosen. $\mathcal{B}$ simulates $\mathcal{A}$ in the following way (where any query to a random oracle is answered randomly, if nothing else is specified). The simulation of the h-oracle is performed as followed, after having chosen a random index $i \in \{1, \ldots, q_H\}$ and initialized a counter j to be 0. The index i will designate the critical query we expect to be involved in the forgery:

- For any new query $(ID_S^* \| ID_R \| m \| r)$ asked to h (by the adversary or by our simulation of `Signcrypt`—see below), we increment the counter j. If $j \neq i$, a new valid message–signature pair (M, S) is taken from the list.
- Then, one chooses a random ρ, defines $h(ID_S \| ID_R \| m \| r) \leftarrow \rho$, and sets

$$s_1 \leftarrow \rho \| r + m \bmod p \quad s_2 \leftarrow \rho \| r + 2m \bmod p \quad r_1 \leftarrow s_1 \oplus f(s_2)$$

- One eventually defines $g(r_1) \leftarrow s_2 \oplus M$, which is a random value, since M is randomly distributed. (Note that for the i-th query to h, we use the designated message μ instead of M, expecting it to be involved in the forgery.) It may fail if $g(r_1)$ has already been defined. However, because of the *fresh* random choice of ρ, this occurs with a probability at most $q_G/2^{k_2}$ for each fresh h-query.

For the i-th query to h, one simply chooses a random output.

In other words, any query m by $\mathcal{A}$ to the oracle `Signcrypt` can be simulated, thanks to the above simulation of h (except for the i-th query to h). Indeed, for answering a `Signcrypt`-query m from ID_S^* to ID_R, one simply chooses a random r, asks for $h(ID_S^* \| ID_R \| m \| r)$, using the above simulation. Except for the i-th query to h, the signature S involved in the pair (M, S) used for the h simulation is a signature c_2 of $r_2 = M$. Using the public key of the receiver ID_R, one can encrypt r_1 in order to obtain c_1. The pair (c_1, c_2) is a valid ciphertext of m. If there is a signcrypt query related to the i-th h-query, we are stuck, we then simply stop the simulation: this i-th query cannot be involved in the forgery, because of the determinism of the process.

Finally, the adversary $\mathcal{A}$ produces a new ciphertext (c_1, c_2), from ID_S^* to ID_R^*, which is valid with the probability greater than $Adv_{Signcrypt}^{UF-CMA}(k)$, unless the above simulation of h fails when trying to assign $h(ID_S \| ID_R \| m \| r) \leftarrow \rho$. Such a failure happens with the probability upper bounded by $q_H q_G/2^{k_2}$. Simulation indeed fails if it fails for any of the messages (the number of messages is q_H).

As we have already mentioned, if a forgery is not related to a specific h-oracle query, then the probability of success is $1/2^{k_2}$. Hence, with probability at least $Adv_{Signcrypt}^{UF-CMA}(k) - (q_G q_H + 1)/2^{k_2}$, we have that the forgery is related to a specific h-oracle query. With the probability $1/q_H$, otherwise we abort the simulation, this ciphertext is involved in the i-th query to the h-oracle and consequently c_2 is a valid signature of μ. Either this is a new signature or it was already involved in a ciphertext (c_1', c_2') to ID_R' produced by `Signcrypt`. In the latter case, since $c_2 = c_2'$, this implies that $c_1 \neq c_1'$ or $ID_R^* \neq ID_R'$. If $ID_R' = ID_R^*$, then $c_1' \neq c_1$ and, thus, because of the determinism of the encryption scheme, it means that $u_1 \neq u_1'$, and then the redundancy may hold with the probability at most $1/2^{k_2}$. If $ID_R \neq ID_R^*$, then again the redundancy may hold with the probability at most $1/2^{k_2}$.

Finally, the probability for $\mathcal{B}$ to produce a new valid signature of μ is greater than

$$Adv_{Sign}^{uUF-RMA}(k) \geq \frac{1}{q_H} \times \left(Adv_{Signcrypt}^{UF-CMA}(k) - \frac{q_G q_H + 2}{2^{k_2}} \right)$$

Furthermore, one can easily see that $q_G = q_g + q_H$, where $q_H \leq q_h + q_{sc}$. $\square$

Proof (of Lemma 9.4) As we are dealing with the insider security model FSO/FUO-IND-CCA2 in the multi-user setting, the adversary has a target receiver ID_R^* in mind. The adversary knows the receiver public key pk_R and has access to the `Unsigncrypt` oracle under sk_R. Further, we assume that an adversary $\mathcal{A}$ observed q_{usc} queries to the `Unsigncrypt` oracle. $\mathcal{A}$ also has chosen a pair of messages m_0 and m_1 and a key pair (sk_S, pk_S) for ID_S. It receives a ciphertext (c_1^*, c_2^*) under (sk_S, pk_R) of either m_0 or m_1. The unknown message is denoted by m_b, where b is the bit the adversary wishes to find out. The adversary $\mathcal{A}$ outputs a bit d which is equal to b with the advantage ε, i.e., $\Pr[d = b] = 1/2 + \varepsilon$. In the following, we use a * for all the internal values used in computing the challenge signcryption.

Let us first remark that because of the randomness of the random oracles f and g, and since $r_1^* \leftarrow s_1^* \oplus f(s_2^*)$ and $r_2^* \leftarrow s_2^* \oplus g(r_1^*)$, to get any information about the bit b (and thus about the encrypted and signed message), the adversary must have got some information about the internal values s_1^* and s_2^* from either the ciphertext or from the plaintext. The former case is only possible if the adversary asks for r_1^* to the oracle g (otherwise it has no information about either s_2^* or s_1^* and thus has no information about the polynomial F, and consequently no information about r^*). The event that the adversary $\mathcal{A}$ has asked the oracle g for r_1^* is denoted by AskG. The latter case means that the adversary has asked either $h(ID_S^* \| ID_R^* \| m_0 \| r^*)$ or $h(ID_S^* \| ID_R^* \| m_1 \| r^*)$. This event is denoted by AskR. Consequently, $\Pr[d = b \mid \neg(\textsc{AskG} \vee \textsc{AskR})] = 1/2$, and thus

$$\begin{aligned}
\varepsilon &= Adv_{Signcrypt}^{IND-CCA2}(k) \\
&= \Pr[d = b] - 1/2 \\
&= \Pr[d = b \wedge (\textsc{AskG} \vee \textsc{AskR})] + \Pr[d = b \wedge \neg(\textsc{AskG} \vee \textsc{AskR})] - 1/2 \\
&= \Pr[d = b \mid \textsc{AskG} \vee \textsc{AskR}] \cdot \Pr[\textsc{AskG} \vee \textsc{AskR}] \\
&\qquad + \Pr[d = b \mid \neg(\textsc{AskG} \vee \textsc{AskR})] \cdot \Pr[\neg(\textsc{AskG} \vee \textsc{AskR})] - 1/2 \\
&\leq \Pr[\textsc{AskG} \vee \textsc{AskR}] + \Pr[\neg(\textsc{AskG} \vee \textsc{AskR})]/2 - 1/2 \\
&\leq \Pr[\textsc{AskG} \vee \textsc{AskR}] \\
&\leq \Pr[\textsc{AskG}] + \Pr[\textsc{AskR} \mid \neg\textsc{AskG}] \\
&\leq \Pr[\textsc{AskG}] + \frac{q_h}{2^{k_1}}
\end{aligned}$$

This last inequality comes from the fact that if AskG does not occur then the adversary has no knowledge of r_1^* and so AskR can only occur by guessing this value.

If AskG occurs, then the plaintext r_1^* of c_1^* has to appear in the queries asked to g. For each query asked to g, one runs the deterministic encryption algorithm and therefore can find the plaintext of a given c_1^*. So we may use the adversary $\mathcal{A}$ to break OW-CPA of the encryption scheme (`EncKeyGen`, `Encrypt`, `Decrypt`). To complete the proof, we have to show how we can simulate all the oracles available to the adversary $\mathcal{A}$. We thus design a simulator $\mathcal{B}$ which receives the private/public keys (sk_S, pk_S) for the signature scheme, from the adversary $\mathcal{A}$, and it is also given the public key pk_R of the encryption scheme. The simulator $\mathcal{B}$ works as follows:

- $\mathcal{B}$ is given a ciphertext c^* (of a random message) to decrypt under the encryption scheme $\mathcal{E}$ and then runs $\mathcal{A}$.
- When $\mathcal{B}$ receives the pair of messages m_0 and m_1 from $\mathcal{A}$, it sets $c_1^* \leftarrow c^*$ and randomly chooses r_2^* that it can sign using the private key of the signature scheme to produce c_2^*. It therefore sends the pair (c_1^*, c_2^*) as a ciphertext of m_b (for some bit b). Finally, the adversary $\mathcal{A}$ follows the attack in which it cannot detect the above simulation of the challenge from a real challenge unless the event AskG happens, which breaks OW-CPA.
- Before simulating the oracle `Unsigncrypt`, let us explain how one deals with h-queries. Indeed, a list Λ_h is managed. For any query $h(ID_S\|ID_R\|m\|r)$, one anticipates the signcryption:

  $$H = h(ID_S\|ID_R\|m\|r) \quad a_0 = H\|r \quad t_1 = a_0 + m \bmod p \quad t_2 = a_0 + 2m \bmod p$$

  Then, $u_1 = t_1 \oplus f(t_2)$ and $u_2 = t_2 \oplus g(u_1)$ (using the canonical simulations of f and g, which are new random elements for new queries). Eventually, one stores $(m, r, H, u_1, u_2, t_1, t_2)$ in the list Λ_h.
- Any call by $\mathcal{A}$ to the oracle `Unsigncrypt` under pk_R can be simulated using the queries–answers of the random oracles. Indeed, to a query (c_1, c_2), one first gets u_2 from c_2, thanks to the public key of the signature scheme ($u_2 =$ `Verify`(pk_S, c_2)). Then, one looks up into Λ_h for tuples $(m, r, H, u_1, u_2, t_1, t_2)$. Then, one checks whether one of the u_1 is really encrypted in c_1 under pk_R, thanks to the deterministic property of the encryption. If no tuple is found, the simulator outputs $\perp$, considering it is a wrong ciphertext. Otherwise, the simulator returns m as the plaintext.

For all the ciphertexts correctly constructed (with $s_2 = t_2$ asked to f, $r_1 = u_1$ asked to g and $(ID_S\|ID_R^*\|m\|r)$ asked to h), the simulation gets back the message. However, the adversary may produce a valid ciphertext without asking $h(ID_S\|ID_R^*\|m\|r)$ required by the above simulation. In that sole case, the simulation may not be perfect.

First, let us assume that $(ID_S\|ID_R^*\|m\|r)$ has not been asked to h:

- If $(ID_S\|ID_R^*\|m\|r) \neq (ID_S^*\|ID_R^*\|m_b\|r^*)$ (the tuple involved in the challenge ciphertext) then $H \leftarrow h(ID_S\|ID_R^*\|m\|r)$ is totally random. The probability that $H\|r$ is equal to a_0 is less than 2^{-k_2} and so $\perp$ is the correct response except with probability 2^{-k_2}.
- In the case where $(ID_S\|ID_R^*\|m\|r) = (ID_S^*\|ID_R^*\|m_b\|r^*)$, since the process to produce r_1 and r_2 is deterministic, $r_1 = r_1^*$ and $r_2 = r_2^*$, the same as in the challenge ciphertext. We remind that both the encryption scheme and the signature scheme are deterministic. Then $c_1 = c_1^*$ and $c_2 = c_2^*$, which is not possible.

Therefore, the probability that the simulation wrongly rejects a valid ciphertext is less than 2^{-k_2}.

If all the decryption simulations are correct (no occurrence of the event $\textsc{BadD}$), we have seen that with a good probability the plaintext c_1^*, and thus of c^*, appears in the queries asked to g, which is immediately detected thanks to the deterministic property of the encryption scheme so

$$\Pr[\textsc{AskG} \mid \neg\textsc{BadD}] \geq \Pr[\textsc{AskG}] - \Pr[\textsc{BadD}] \geq \left(\varepsilon - \frac{q_h}{2^{k_1}}\right) - \frac{q_{usc}}{2^{k_2}}$$

$\square$