Deniable Authentication and Key Exchange

Mario Di Raimondo [*], Rosario Gennaro [**], and Hugo Krawczyk [***]

Abstract. We extend the definitional work of Dwork, Naor and Sahai from deniable authentication to deniable key-exchange protocols. We then use these definitions to prove the deniability features of SKEME and SIGMA, two natural and efficient protocols which serve as basis for the Internet Key Exchange (IKE) protocol. The two protocols require distinct approaches to their deniability analysis, hence highlighting important definitional issues as well as necessitating different tools in the analysis.

SKEME is an encryption-based protocol for which we prove full deniability based on the plaintext awareness of the underlying encryption scheme. Interestingly SKEME's deniability is possibly the first "natural" application which essentially requires plaintext awareness (until now this notion has been mainly used as a tool for proving chosen-ciphertext security); in particular this use of plaintext awareness is not tied to the random oracle model. SIGMA, on the other hand, uses non-repudiable signatures for authentication and hence cannot be proven to be fully deniable. Yet we are able to prove a weaker, but meaningful, "partial deniability" property: a party may not be able to deny that it was "alive" at some point in time but can fully deny the contents of its communications and the identity of its interlocutors.

We remark that the deniability of SKEME and SIGMA holds in a *concurrent* setting and does not essentially rely on the random oracle model.

Keywords: Key Exchange, Authentication, Deniability, Privacy, Zero-Knowledge, Plaintext-Awareness.

[*] Dipartimento di Matematica ed Informatica – Università di Catania, Italy. diraimondo@dmi.unict.it

[**] IBM T.J.Watson Research Center, USA. rosario@watson.ibm.com

[***] IBM T.J.Watson Research Center, USA. hugo@ee.technion.ac.il

1 Introduction

Privacy of communications has been the main object of study in cryptography for centuries. Its classical goal: prevent unauthorized parties from accessing secret or confidential information. In this setting the focus is on establishing authenticated and secret communication (a.k.a. "secure channels") with authorized peers. The intent is to defend the communications from an "unauthorized third party" in the form of an eavesdropper or active attacker; there is no attempt at preventing an authorized peer from disclosing information it receives legitimately. Today, with the transfer of our personal, social, economic and political lives to digital form, privacy has become a much wider and central notion. Modern cryptography recognized these issues early on through anonymity-related notions [14, 15], mix networks [13], undeniable signatures [16], private information retrieval [18], and more. More recently, we have seen a huge increase in the treatment of broader privacy issues in the crypto/security community (e.g. [38, 26, 17]). This paper focuses on an essential aspect of privacy: the deniability of every-day digital communications.

Deniable communication has always been a central concern in personal and business communications, with off-the-record communication serving as an essential social and political tool. Given that much of these interactions now happen over digital media (email, instant messaging, web transactions, virtual private networks) it is of central importance to provide these communications with "off-the-record" or deniability capabilities: the author or sender of a message should be able to deny, e.g., in court, that he or she sent that message. (Needless to say, there are special applications where non-repudiation is essential, but this is not the case for most of our communications.) One of the challenges of deniability in the digital world is that deniability is at odds, at least without careful design, with remote authentication. That is, Alice needs to be able to get a digital proof that she is talking to Bob but that proof should not leave any trace that will convince a third party that Bob talked (or said something specific) to Alice. This should be the case even if Alice herself is trying to prove the existence of the conversation to such third party! Thus, while in the traditional "secure channels" setting one is not defending against misbehavior by the (authorized) peer to the communication, in the deniability setting the potential attackers include the authorized (identified and authenticated) peer.

The first to formally treat the deniable authentication problem were Dwork, Naor and Sahai in [27], followed by a series of papers including [41, 42, 34, 23]. On the more applied front, a practical (and widespread) protocol that set "plausible deniability" as a desirable property (though, not as an essential goal) is the IKE protocol [31]; in particular, this motivated the design of the SKEME protocol [36] that became part of IKEv1. More recently, [7, 24] treat explicitly off-the-record communications in the setting of instant-messaging communications.

Our present paper is motivated by the above works. One missing link in these works is the formal treatment of deniability for key-exchange (KE) protocols. Note that when using symmetric shared keys to authenticate/encrypt information then deniability is easy to achieve at least *as long as the secret key cannot* *be tied, via third-party verifiable proofs, to the identities of the peers.* However, when the symmetric keys are established via a KE protocol, which in turn uses public key techniques for authentication (as it is common in today's communications), then the weak link for deniability becomes this KE protocol. If its authentication mechanisms leave a "proof of communication" then deniability is lost.

The following example is useful to illustrate how a KE protocol can indeed leave such a "proof of communication". Here is a simple 3-message variant of an ISO Diffie-Hellman KE protocol [32, 10]:

$$(1)\ A \to B\colon g^x \qquad (2)\ B \to A\colon g^y,\ sig_B(g^x, g^y, \mathrm{id}_A) \qquad (3)\ A \to B\colon sig_A(g^y, g^x, \mathrm{id}_B)$$

The protocol makes use of digital signatures as the most natural (and scalable) tool for remote authentication. In addition, as part of the essential features of a secure KE protocol the identities of the communicating parties (i.e., id_A and id_B) are tied to the exchanged key via these signatures. However, by signing the peer's identity each of the parties of the protocol leaves an undeniable proof of communication between these parties. Note that in this case the non-deniability of the protocol is particularly serious: not only can A prove that B talked to her, but even an eavesdropper that obtains these signatures will be able to provide such a proof. Unfortunately, if one omits these identities in the signatures the protocol becomes completely insecure [28].

This example serves to stress the conflict between authentication and deniability; and, in the case of KE, the conflict between deniability and the binding between identities and the exchanged key so central to KE security. One of our central contributions is in showing that carefully designed protocols may provide for sound KE security as well as for significant levels of deniability. Fortunately, we show this to be the case for some well-known and practical KE protocols.

Defining Deniability. The notion of *deniability* in public key authentication formalized by Dwork, Naor and Sahai [27] using the *simulation* paradigm. Informally, a protocol is deniable if the view of any receiver (or verifier) can be simulated by a machine that does not know the secret key of the sender (or prover). The idea behind this natural and appealing definition is that the transcript of the protocol owned by the receiver, cannot be used to trace this conversation back to a specific sender, since the receiver could have produced it via the simulator machine.

Thus, the basic property of a deniable protocol follows the notion of *zero-knowledge* [29]. However, for deniability one needs a stronger simulator than in the case of ZK: while in ZK the simulator is basically a "mental experiment", for deniability, as pointed out by Pass [42], the simulator must be a real machine that works in the real world. For example, in a model in which there is a common reference string (CRS), the ZK definition allows the simulator to produce the CRS together with some extra information that will allow it to produce simulations of the actual protocols. On the other hand, such a simulator would not necessarily prove deniability because in real life the CRS is fixed and cannot be modified by the verifier (similar considerations apply to proofs in the "random oracle model"). Another challenge in dealing with ZK-based proofs is that those usually make use of "rewinding". As pointed out in [27], this technique significantly limits the applicability of the proof in a real-life concurrent-executions setting.

Deniability for Key Exchange Protocols. In a KE protocol, two parties engage in a protocol whose result is a (session) key K which only the two of them know, and they are assured to be sharing K with each other. They will use K to encrypt and authenticate messages in the session, using a symmetric-key authentication mechanism that is deniable *provided that the key cannot be traced to either party.*

Thus, the most important component for the deniability of electronic communications is the deniability of KE protocols. If the parties can deny having exchanged a key with the other party, then the rest of the communication can also be denied.[1]

Our contributions. After recalling the definition of deniable authentication from [27] we propose a definition of deniable key exchange protocols, which still adheres to the simulation paradigm. The extension is not trivial as we are moving to a protocol which outputs a secret value (rather than the simple accept/reject bit of an authentication protocol). In particular KE deniability requires the simulation not only of the entire transcript, but of the output key as well, since the key will be passed to an arbitrary security protocol after the exchange phase is completed.

We then study the deniability of SKEME [36] and SIGMA [37], two practical KE protocols which form the cryptographic core of IKE, the Internet Key Exchange of IPSEC [31, 35].

For SKEME we show the strong form of deniability guaranteed by our definition. The analysis has several interesting features; in particular we first abstract out a basic message authentication mechanism from the key exchange protocol and prove its deniability in the setting of Dwork et al. [27]. We then use this result to show the deniability of the full SKEME protocol.

In the case of SKEME the authentication protocol is very simple: the sender has an encryption public/private key pair $(\mathsf{pk},\mathsf{sk})$ and the receiver who wants the message m authenticated, sends $\mathsf{enc}_{\mathsf{pk}}(k)$ where k is a random key. The sender responds with $\mathsf{MAC}_k(m)$. This authenticator, derived from [36], has been proven secure in [1] provided that the encryption function is CCA2-secure [25]. (A related authenticator and proof appears in [25, 27].) But what about deniability? This two-message scheme was not known to be deniable, and some evidence (including the low number of messages) indicated that it may not be deniable. We show that CCA2-security for enc is *not* sufficient for deniability by showing that there is a CCA2-secure encryption scheme for which the above protocol is not deniable. We then show that deniability holds if the encryption scheme is *plaintext-aware* (either in the standard model [4, 21] or in the random oracle

[1] Needless to say, we limit ourselves to avoiding "algorithmic" proofs of communication, and ignore other means that courts, or other entities, may accept as evidence such as physical tapping (or just the word of a gentleman...).

model [6, 2]) Interestingly this makes deniability one of the first applications to essentially require plaintext awareness; so far plaintext awareness was mainly used as a tool to prove CCA2 security.

The case of SIGMA is more involved. Since SIGMA uses signature-based authentication, deniability (defined as full simulation) cannot be achieved. However, in spite of its use of signatures (and in contrast to the above ISO example) SIGMA offers some valuable deniability properties. We capture these properties via a modified notion of *partial deniability* for key-exchange protocols, in which a party can deny the identity of the parties he or she exchanges keys with, as well as the content of the subsequent communications protected by those keys. Based on this, we show the 4-message variant of SIGMA (known as SIGMA-R) to be partially deniable. The result holds under a "key awareness" notion, which can be plaussibly assumed to hold for "natural" MAC and hashing functions (and can formally be shown to hold in the random oracle model).

Concurrency An important feature is that our proof of deniability for SKEME and SIGMA holds in a *concurrent* setting, where the adversary can open and schedule sessions in an arbitrary way [27]. This is of utmost importance for practical applications run in an open network like the Internet. In addition, our notions and proofs, while meaningful in the random oracle model, do *not* essentially depend on it.

Related Work. As we said earlier deniable authentication has been studied from both the theoretical [27, 42, 41, 34, 23] and the practical [36, 31, 8, 40, 7, 24] points of view. For the case of key exchange protocols, where both parties have registered public keys there are other approaches to deniable authentication: *Designated Verifier Proofs* [33] permit to create signatures that convince only the intended recipient (using his public key); *Ring Signatures* [44] permit a member of an *ad hoc* group to sign a message on behalf of the group, i.e. it is impossible to trace the actual signer inside the group. This solution can be used to create deniable signatures by choosing the sender and the receiver as members of the group: the signature is deniable as the receiver could have created it too . Our approach is different in that our goal is to prove deniability for real-life protocols used in practice (which do not use the above tools).

Formalisms of KE security has been extensively studied [1, 45, 10, 12] yet none of those studies considered deniability explicitly and/or formally. Informal treatment of deniability issues for KE protocols can be found in [36, 7, 8, 40, 24].

Finally we recall the notion of *deniable encryption* [9] which allow a sender to encrypt a message m in a ciphertext c, under the public key of a receiver, while allowing him to later *deny* what the content of the ciphertext is. Deniable encryption is also motivated by privacy-preserving applications (such as electronic voting), but the notion and the technique are unrelated to the problem we are tackling (denying that an interaction between two parties ever took place).

2 Definition of Deniable Key Exchange

Our presentation and definition of the notions of deniability in this section follows essentially the approach and definition from Dwork, Naor and Sahai [27], the first to formalize the deniability of authentication protocols. For lack of space, the notion of authentication protocol from [27] is presented in Appendix A.1. Next we recall their definition of deniable authentication that we use as the basis for formalizing deniability of key exchange protocols. (Throughout the paper we will use the standard polynomial-time complexity notions, such as indistinguishability, negligible probabilities, etc.; in particular, all algorithms and machines are probabilistic polynomial-time.)

Deniable Authentication. Intuitively an authentication protocol is deniable if a (possibly dishonest) receiver cannot convince a third party (let's call it, the *judge*) that a given sender S authenticated a given message m. This notion was first formalized in [27] using a zero-knowledge formalism. The idea is that an authentication protocol is deniable if the receiver's *view* of the protocol can be *simulated* by an efficient machine (called the *simulator*) that does *not* know the secret key of the sender S. In other words, the receiver interacting with the simulator obtains views with the same distribution than when interacting with the real sender S. Thus, when the receiver (the attacker in this setting) brings such a view to the judge, this view will not be a convincing evidence of the interaction with S since the same view could have been generated by the receiver alone by running the simulator. We recall the formal definition next.

Consider an adversary $\mathcal{M}$ (for "malicious") acting as a receiver on input pk. The adversary may also have some auxiliary input aux taken from a distribution AUX of such inputs. This auxiliary input models some extra information that the adversary might have gathered in some other form; for example, if $\mathcal{M}$ has been eavesdropping on correctly executed protocols between other parties and S, AUX will consist of legal transcripts of runs of the authentication protocol.

The adversary $\mathcal{M}$ starts a *concurrent interaction* (as defined in [27]) with the sender S. In other words, $\mathcal{M}$ starts an arbitrary number of executions of the authentication protocol with S with public key pk, choosing the input messages for these executions. These executions can be arbitrarily scheduled and interleaved by $\mathcal{M}$. The adversary's *view* of this interaction is then defined as the transcript of the full interaction between $\mathcal{M}$ and S, together with the internal coin tosses of $\mathcal{M}$. We denote this as $\mathsf{View}^{\mathsf{S}}_{\mathcal{M}}(\mathsf{pk}, aux)$.

Definition 1. [27] *We say that (*AKG*,*S*,*R*) is* concurrently deniable *with respect to the class AUX of auxiliary inputs if for any adversary $\mathcal{M}$, acting as the receiver on input* pk *and any auxiliary input $aux \in AUX$, there exists a simulator $SIM^{S}_{\mathcal{M}}$ that, running on the same inputs, produces a simulated view which is indistinguishable from the real one. In other words, consider these two probability distributions:*

$$\mathcal{R}eal(n, aux) = [(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{AKG}(1^n)\ ;\ (aux, \mathsf{pk}, \mathsf{View}^{\mathsf{S}}_{\mathcal{M}}(\mathsf{pk}, aux))]$$

$$\mathcal{S}im(n, aux) = [(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{AKG}(1^n)\ ;\ (aux, \mathsf{pk}, SIM^{\mathsf{S}}_{\mathcal{M}}(\mathsf{pk}, aux))]$$

then for all probabilistic polynomial-time machine Dist *and all* $aux \in AUX$

$$|Pr_{x\in\mathcal{R}eal(n,aux)}[\mathsf{Dist}(x)=1] - Pr_{x\in\mathcal{S}im(n,aux)}[\mathsf{Dist}(x)=1]| \leq \mathsf{negl}(n)$$

We stress that the the simulator has *all* the same inputs as $\mathcal{M}$, including its random coins (alternatively assume that the simulator provides these coins to $\mathcal{M}$).

Remark (*Off-line vs. on-line judges.*) In the deniability context (also in the case of KE protocols) the *distinguisher* Dist represents the role of the *judge* which needs to decide if the transcript presented by $\mathcal{M}$ corresponds to a real execution of the protocol with S or to a simulated view of such run. This distinguisher is presented with the transcript as well as with the inputs of $\mathcal{M}$ including the auxiliary input *aux* (which are also the inputs on which *SIM* is run). Hence, this formulation corresponds to the situation in which the transcript is generated without the judge intervention, i.e., the judge is invoked *a posteriori* to decide if the message m was really authenticated by S or not. This formulation is in line with the standard zero-knowledge definitions and also with the intuitive notion of deniability. Yet, one could contemplate a stronger setting in which the judge is allowed to interact with the adversary *before* the authentication protocol takes place or even *during* the run of the protocol between $\mathcal{M}$ and S. In the latter case, there is little one can do to provide deniability with respect to this "on-line" judge since the judge itself can run the protocol directly with S (or using $\mathcal{M}$ as a relaying intermediary) and be convinced by this direct interaction as any other receiver R. In the case of interaction between the judge and adversary $\mathcal{M}$ before the (alleged) run between $\mathcal{M}$ and S, but not during the run, the above definition per se is not sufficient to ensure deniability. Yet, some protocols will achieve some form of deniability also in this case. We will not formalize this stronger notion here but when presenting specific protocols we will discuss the extent to which they achieve this stronger notion.

Deniable Key Exchange. Here we extend the above definition of deniability to the setting of (authenticated) key-exchange (KE) protocols. As in the case of authentication, we present a simplified definition of a KE protocol as a stand-alone procedure, and then define the deniability property in a concurrent-execution setting. A more general treatment of the subject, including a full integration of deniability with a model of KE security in a multi-party setting is left for future work. By simplifying the formal treatment in our presentation here, and focusing on the concurrent setting, we are able to highlight the technical and conceptual issues raised in the investigation of deniability. Also, by studying the deniability features of specific KE protocols that were proven secure in the literature we can build on these works and concentrate mainly on the novelty of the deniability aspects. For formal definitions of KE security see [5, 45, 10, 11].

In a key exchange protocol, two parties, say A and B, are associated with public keys pk_A and pk_B respectively, for which they each own the matching secret key sk_A and sk_B. We assume that public/secret keys are generated according to a key generation algorithm KG which is part of the specification of the KE protocol, and these are used in the authentication steps of the KE protocol. The protocol specifies the interaction between A and B (one acting as "initiator" and the other as "responder") and whose result is either a (session) key K or "error" if some of the operations/verifications in the protocol fail. The basic (and simplified) security requirement in a KE protocol is that if A outputs a session key K and associates it to peer B then the only party that may possibly know K is B; and if B outputs the same session key then it associates it to peer A. Note however that this security guarantee is provided only for sessions (i.e., runs of the KE protocol) in which both peers are uncorrupted.

In great *contrast,* the deniability guarantee of a KE session is most relevant when one of the peers is dishonest (and the other honest). The goal of deniability is to prevent either A or B from proving to a judge that they exchanged a key with a specific party, and to prevent a proof of what the contents of a communication protected with that key were. Once again we model deniability via simulation along the lines of Definition 1 but with some important differences, arising mainly from the fact that not only is the KE protocol itself that needs deniability but also the communications that use the resultant session key.

Let Σ be a key-exchange protocol defined by a key generation algorithm KG and interactive machines Σ_I, Σ_R specifying the roles of the (honest) initiator (the party who sends the first message) and responder, respectively. Both Σ_I and Σ_R run on input their own secret and public keys and, typically, also on input the identity and public key of a specified peer (in some cases, however, the identity and public key of the peer are not provided as input at the onset of the interaction but these values are rather learned during the run of the protocol [12]). Each run of the KE protocol by a party is called a session. Upon completion, the protocol outputs either error (e.g., an authentication operation failed) or outputs a session key.

Consider an adversary $\mathcal{M}$ which runs on input an arbitrary number of public keys $\mathbf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$, randomly chosen according to KG and associated with the honest users in the network. $\mathcal{M}$ also has an auxiliary input aux as in Definition 1. The adversary initiates an arbitrary number of executions of Σ with the honest parties, some as an initiator, others as a responder[2]. The executions are concurrent, i.e. scheduled and interleaved arbitrarily by the adversary. The view of $\mathcal{M}$ consists of its internal coin tosses, the transcript of the entire interaction *and the session keys computed in all the protocols in which $\mathcal{M}$ participated* (if the session does not complete, the session key is defined as an error value). We denote this view as $\mathsf{View}_{\mathcal{M}}(\mathbf{pk}, aux)$.

The definition below follows the traditional approach of simulation of the adversary's view. However, it is important to highlight an element in this definition that differentiates it essentially from deniability in the message authentication setting of the previous subsection: the inclusion of the computed secret key to the adversary's view. Recall that the goal of deniability in a KE protocol is not only to prevent a (adversarial) party $\mathcal{M}$ from proving that another (honest) party B talked to $\mathcal{M}$ but also to prevent $\mathcal{M}$ from proving to a third party the contents

[2] In these executions the adversary will use public keys which she can select arbitrarily, see the Remark after the definition.

of a communication in which B participated. Since KE protocols are typically run in order to agree on a session key which is later used to authenticate further communication, *it is essential that not only the communication during the KE session be simulatable but also the value of the session key should be part of the output of the simulation.* In this case, when an attacker brings to a "judge" evidence against B in the form of a key exchange or subsequent authenticated information, the simulatability of the exchange and the resultant key will make this evidence worthless.

A technical point worth noting is that the simulator is required to output the session key only in case that the simulated (honest) party completes the protocol with a session key as output. Only in this case the party will pass the session key for use in some application, and hence it is then when the key needs deniability. Also, the reason we require that the whole session key be simulated (rather than, say, the ability to compute a MAC value with that key) is that we do not know what applications and in which way the key will be used (e.g., some applications may transmit the whole key in the clear, in which case anything short of full key simulatability will be insufficient).

Definition 2. *We say that* $(\mathsf{KG}, \Sigma_I, \Sigma_R)$ *is a* concurrently deniable *key exchange protocol with respect to the class* AUX *of auxiliary inputs if for any adversary* $\mathcal{M}$*, for any input of public keys* $\mathbf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$ *and any auxiliary input* $aux \in AUX$*, there exists a simulator* $SIM_{\mathcal{M}}$ *that, running on the same inputs as* $\mathcal{M}$*, produces a simulated view which is indistinguishable from the real view of* $\mathcal{M}$*.*

That is, consider the following two probability distributions where $\mathbf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$ *is the set of public keys of the honest parties:*

$$\mathcal{R}eal(n, aux) = [(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KG}(1^n)\ ;\ (aux, \mathbf{pk}, \mathsf{View}_{\mathcal{M}}(\mathbf{pk}, aux)]$$

$$\mathcal{S}im(n, aux) = [(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KG}(1^n)\ ;\ (aux, \mathbf{pk}, SIM_{\mathcal{M}}(\mathbf{pk}, aux)]$$

then for all probabilistic poly-time machines Dist *and all* $aux \in AUX$

$$|Pr_{x \in \mathcal{R}eal(n,aux)}[\mathsf{Dist}(x) = 1] - Pr_{x \in \mathcal{S}im(n,aux)}[\mathsf{Dist}(x) = 1]| \leq \mathsf{negl}(n)$$

Remarks on the definition. First note that impersonation is not the issue here: when $\mathcal{M}$ is interacting with B she is not trying to impersonate A but rather, she is trying to obtain a proof that she herself interacted with B and established a key with him. The goal of deniability is to prevent $\mathcal{M}$ from proving to a third party that this was the case. Thus, when $\mathcal{M}$ interacts with B we can assume (wlog) that she will do so using a public key $\mathsf{pk}_{\mathcal{M}}$ (which may or may not be associated with $\mathcal{M}$'s identity). Indeed, since our definition guarantees that even when the attacker $\mathcal{M}$ runs the key generation algorithm to generate a public key (thus knowing the corresponding secret key) she cannot prove that B talked to her, then $\mathcal{M}$ can certainly not be able to prove that B talked to any *other* party A (in particular, this implies deniability with respect to eavesdroppers). In addition, while $\mathcal{M}$ may decide to reveal her secret key $\mathsf{sk}_{\mathcal{M}}$ to help in proving that B talked to her, she does not have to do so (actually $\mathcal{M}$ may use a public key for which she does not know a corresponding private key!).

In Section 4 we present a relaxed variant of Definition 2 which provides for a weaker, yet meaningful, notion of (limited) deniability.

3 Deniability of SKEME

In this section we prove the deniability of the key exchange protocol SKEME [36] which uses public key encryption as a means of authentication. First we abstract out the authentication protocol which is at the core of SKEME and prove that it is deniable if the encryption scheme used is plaintext-aware according to the definition in [4]. We also show that chosen-ciphertext security is not sufficient to prove deniability. Then we use this result to prove the deniability of the SKEME key exchange protocol[3]. In the following we denote an encryption scheme $\mathcal{E} = (\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$ which are the key generation, encryption and decryption algorithms, respectively. We recall the traditional notions of security for encryption in Appendix A.2.

3.1 Encryption-based Deniable Authentication

Here we study the authentication protocol based on public-key encryption which is at the core of SKEME [36] (related protocols are studied in [25, 27]).

Let pk be the sender's S public key and sk the related secret key, for a public key encryption scheme $\mathcal{E}$. The receiver chooses a random key k and encrypts it under the sender's public key as $c = \mathsf{enc}(\mathsf{pk}, k)$. The sender decrypts such key as $k = \mathsf{dec}(\mathsf{sk}, c)$ and uses it to create a valid MAC of the message m under the key k. If the decryption is invalid, the sender chooses a random key k' and sends a MAC on m computed with this random key. The protocol is described in Figure 1.

The receiver's belief that S is really authenticating m comes from the fact that she is the only one able to decrypt k. Indeed this authentication protocol is proven secure in [1] if $\mathcal{E}$ is secure against adaptive chosen-ciphertext attack (IND-CCA2). On the other hand, the receiver could create the whole transcript on his own, so that the authentication seems intuitively to be deniable.

In fact, this scheme is perfectly deniable against an honest receiver, since the simulator SIM_{R} can easily produce valid transcripts on his own. What happens against a dishonest receiver $\mathcal{M}$? To formally prove the deniability we need to create a simulator $SIM_{\mathcal{M}}$ that is able to produce valid transcripts interacting with the malicious receiver $\mathcal{M}$. When the simulator receives the encrypted challenge $c = \mathsf{enc}(\mathsf{pk}, k)$, how are we going to simulate the reply $t = \mathsf{MAC}_k(m)$? The simulator doesn't know the sender's secret key sk!

[3] Interestingly, this appears to be the first "essential" application of plaintext-awareness; before, this notion was used mainly as a tool to prove chosen-ciphertext security.

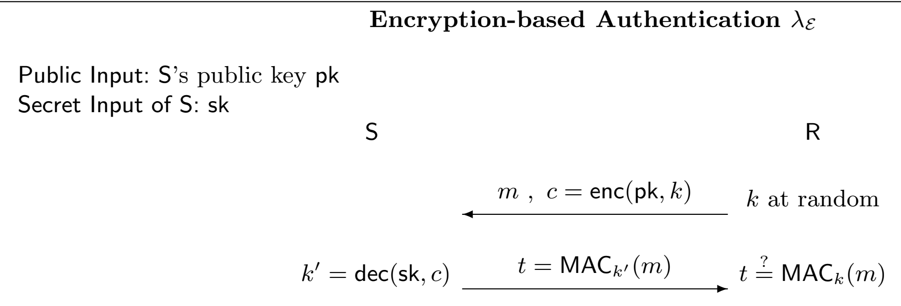
Encryption-based Authentication $\lambda_{\mathcal{E}}$
Public Input: S's public key pk
Secret Input of S: sk
S
R
$m\ ,\ c = \mathsf{enc}(\mathsf{pk}, k)$
k at random
$k' = \mathsf{dec}(\mathsf{sk}, c)$
$t = \mathsf{MAC}_{k'}(m)$
$t \overset{?}{=} \mathsf{MAC}_k(m)$

Fig. 1. Protocol $\lambda_{\mathcal{E}}$: A method of authentication based on public-key encryption

In order to make the above scheme provably deniable, one can add a challenge-response sub-protocol (in a way similar to the authentication protocol in Dwork *et al.* [27]). Basically, the sender instead of replying with the MAC t replies with a commitment to t. Upon receiving such commitment h, the receiver reveals the key k encrypted in the first message. If this key equals the one that he decrypted, the sender opens the commitment, revealing the MAC t.

This variant is still unforgeable assuming that $\mathcal{E}$ is IND-CCA2. Deniability follows from a standard black-box ZK simulation, which however includes a "rewinding" step for the malicious receiver (the simulator commits to random junk in the second message, waits to see k in the third, rewinds the receiver to the second message where it now places a correct commitment to t). This rewinding step limits the applicability of the protocol in a concurrent setting: in this case the proof of security guarantees deniability only if a logarithmic (in the security parameter) number of sessions are concurrently executed [27].

Adding the extra commitment step, thus (i) modifies the protocol; (ii) adds rounds and (iii) yields a non-concurrent solution. Our first result is to show that under the sole assumption that $\mathcal{E}$ is CCA2-secure, the protocol $\lambda_{\mathcal{E}}$ is not deniable.

Theorem 1. *There exists a CCA2-secure encryption scheme $\mathcal{E}$, such that $\lambda_{\mathcal{E}}$ is* not *deniable.*

We prove the above theorem in Appendix B, by presenting a counter-example: an encryption scheme $\mathcal{E}$ which is CCA2-secure but for which the protocol $\lambda_{\mathcal{E}}$ is not deniable.

In contrast we show next that *concurrent deniability* for $\lambda_{\mathcal{E}}$ can be proven by making a stronger assumption on the underlying encryption scheme, namely *(PA-2) plaintext-awareness* [4]. We recall the formal definition of plaintext-awareness in Appendix A.3; here we give an informal description of this notion which suffices to obtain an intuitive understanding of the proof.

Intuitively, we say that $\mathcal{E}$ is *PA-1 plaintext-aware* if for any adversary $\mathbf{C}$ that on input a public key pk outputs a valid ciphertext c there is a "companion" machine $\mathbf{C}^*$ that outputs the matching plaintext. Think of $\mathbf{C}^*$ as the *alter ego* of $\mathbf{C}$: the definition basically implies that if $\mathbf{C}$ produces a valid ciphertext it must "know" the corresponding plaintext. A strengthening of this notion, PA-2, accommodates the fact that an attacker may have access to a set of ciphertexts computed under public key pk but not produced by the attacker itself; for example, these ciphertexts could have been generated by other, honest, parties in the system communicating with the owner of this public key. Hence, the definition of PA-1 is strengthened so that the above adversary $\mathbf{C}$ is also given as input a list of valid ciphertexts for which it does not know the corresponding plaintexts, and the companion machine $\mathbf{C}^*$ is defined to yield the corresponding plaintext only for valid ciphertexts produced by $\mathbf{C}$ which are not in the above list. The resultant notion is called *PA-2 plaintext-awareness* and its formal definition is recalled in Appendix A.3.

When coming to prove the deniability property of protocol $\lambda_{\mathcal{E}}$, the PA-1 notion is therefore too weak to represent the common situation in which multiple copies of the protocol are run concurrently since in this case the attacker does have access to valid ciphertexts created by other parties. In particular, in the case of protocol $\lambda_{\mathcal{E}}$ the attacker may "replay" such ciphertexts in a communication with the owner of pk without knowing the encrypted plaintext. Our result, therefore, depends on the encryption function being PA-2. In this case we will use the auxiliary input AUX to represent a list of valid transcripts of protocol $\lambda_{\mathcal{E}}$ (with pk as the sender's public key) gathered by the attacker in the network.

We denote by $TR(\mathsf{pk})$ the set of legal transcripts of the protocol $\lambda_{\mathcal{E}}$ under public key pk and let the auxiliary input aux, with respect to which the deniability of $\lambda_{\mathcal{E}}$ is established, be a list of transcripts sampled from $TR(\mathsf{pk})$ and for which the adversary $\mathcal{M}$ has no information on the plaintexts (and randomness) used to generate the ciphertexts included in these transcripts. In other words, we assume these to be transcripts generated by *honest parties* in interaction with S.

Theorem 2. *If the encryption scheme $\mathcal{E}$ is PA-2 and IND-CPA secure, then the protocol $\lambda_{\mathcal{E}}$ is concurrently deniable with respect to the class of auxiliary inputs $TR(\mathsf{pk})$.*

Proof. To show that the protocol $\lambda_{\mathcal{E}}$ is deniable against a malicious receiver $\mathcal{M}$ we construct a simulator $SIM_{\mathcal{M}}$ that interacting with $\mathcal{M}$, creates transcripts that are indistinguishable from the real transcripts between $\mathcal{M}$ and S. The following proof shows a non-rewinding simulator for a single execution of the protocol and hence the simulation for the concurrent execution of the protocol follows by composing the (non-rewinding) simulations.

We first provide the idea of the proof in an informal way. The simulator $SIM_{\mathcal{M}}$ acts as follows: when the attacker $\mathcal{M}$ sends a ciphertext c that appears in the aux list (for which $\mathcal{M}$ does not know the corresponding plaintext), $SIM_{\mathcal{M}}$ responds with a MAC value computed under a random key (in this case the attacker has no access to the real key encrypted in the ciphertext and hence the MAC under a random key is indistinguishable from the real MAC) while if the ciphertext c is not in the list $SIM_{\mathcal{M}}$ will resort to the "alter ego" extractor

$\mathcal{M}^*$ to provide with the plaintext under c; now $SIM_{\mathcal{M}}$ can compute the correct MAC value (if c happens to be an invalid ciphertext, $\mathcal{M}^*$ will provide this information to $SIM_{\mathcal{M}}$ and a random MAC will be used). This results in a simulation that is inherently non black-box and which does not use rewinding. The formal argument follows.

We use the following notation and terminology. Given an adversary $\mathcal{M}$ and a sequence of random coins r, we denote by $\mathcal{M}_1(r)$ the pair (m, c) output by $\mathcal{M}$, using coins r, as the first protocol message. If $\mathcal{M}(r)$ does not produce output (i.e., does not send a message at all) we denote this output as $\perp$. We will assume that if the output of $\mathcal{M}_1$ is not empty then it can be parsed into a pair (m, c) (with a possibly invalid ciphertext c). Now we consider a related adversary, denoted $\mathbf{M}$ defined to have access to a decryption oracle under S's private key and related to $\mathcal{M}$ in the following way. If $\mathcal{M}(r) = \perp$ then $\mathbf{M}(r)$ does not produce output. If $\mathcal{M}_1(r) = (m, c)$ then $\mathbf{M}(r)$ outputs the same pair (m, c) but in addition it queries the decryption oracle on ciphertext c. By assumption, the encryption scheme $\mathcal{E}$ is PA-2 and therefore there exists an extractor $\mathbf{M}^*$ that simulates the decryption oracle for $\mathbf{M}$ without having access to the private key of S. Moreover, we will assume that algorithm $\mathbf{M}$ not only outputs the pair (m, c) as $\mathcal{M}$ does, but also outputs its random coins r. By the PA-2 definition, the simulation of $\mathbf{M}$ by $\mathbf{M}^*$ is indistinguishable also for this specification of $\mathbf{M}$.

In addition, we define L to be the list of ciphertexts included in the transcripts set aux, the auxiliary input to $\mathcal{M}$ (recall that aux includes transcripts of $\lambda_{\mathcal{E}}$ executions generated by honest parties interacting with S and therefore these transcripts include valid ciphertexts for which $\mathcal{M}$ has no information on the encrypted plaintexts or the encryption coins). For compatibility with the formalism in the definition of PA-2 we will consider a "plaintext creator" $\mathbf{P}$ operated by $\mathbf{M}$ such that each query from $\mathbf{M}$ to $\mathbf{P}$ is answered by $\mathbf{P}$ with the next ciphertext in the list L.

Having defined the above algorithms and notation we proceed to describe the simulator $SIM_{\mathcal{M}}$. The simulator runs the attacker $\mathcal{M}$ under a sequence of random coins r chosen (uniformly) by $SIM_{\mathcal{M}}$ itself. If $\mathcal{M}(r)$ does not produce output (i.e., $\mathcal{M}_1(r) = \perp$) then $SIM_{\mathcal{M}}$ does not produce output either. If $\mathcal{M}(r)$ produces a pair (m, c) then $SIM_{\mathcal{M}}$ runs $\mathbf{M}^*(r)$ which will output the pair (m, c) as well as a value p corresponding to the simulated decryption of c. This value p will be $\perp$ in two cases: either c is invalid or $c \in L$. In this case, i.e., $p = \perp$, $SIM_{\mathcal{M}}$ chooses a random MAC key k and sets $t = \mathsf{MAC}_k(m)$. Else p equals (or encodes) a MAC key k and $SIM_{\mathcal{M}}$ sets $t = \mathsf{MAC}_k(m)$ for this value of k. Now $SIM_{\mathcal{M}}$ outputs the transcript:

$$\text{coins} = r,\ \text{msg1} = (m, c),\ \text{msg2} = t.$$

We proceed to show that this distribution generated by $SIM_{\mathcal{M}}$ is indistinguishable from the real transcripts between $\mathcal{M}$ and S. In the case that $p = \perp$ is due to the invalidity of the ciphertext c, both the simulation and the real transcript include a MAC value computed under a random key and hence the two distributions are identical. When $p = \perp$ and $c \in L$, the real transcript will include $t = \mathsf{MAC}_k(m)$ computed under the real key encrypted under c while the simulation includes a MAC value computed under a random key. Yet these two distributions are indistinguishable due to the IND-CCA2 property of the encryption scheme $\mathcal{E}$ (we assume that $\mathcal{E}$ is PA-2 and IND-CPA and hence it is IND-CCA2). Finally, when the pair $\mathcal{M}_1(r)$ includes a valid ciphertext $c \notin L$ then, by the definition of PA-2, the distribution of plaintexts p output by the extractor algorithm $\mathbf{M}^*$ is indistinguishable from the distribution of real plaintexts encrypted under valid ciphertexts c. Therefore the value $\mathsf{MAC}_k(m)$ computed by $SIM_{\mathcal{M}}$ in this case is indistinguishable from the MAC value computed by S in a real interaction with $\mathcal{M}$. Note that the above argument holds also when the coins r of $\mathcal{M}$ are output (as required by the definition of deniability) since, as said, $\mathbf{M}^*$ simulates a decryption oracle correctly also when $\mathbf{M}$ outputs such coins.

3.2 The SKEME Key Exchange Protocol

The SKEME key exchange from [36] is described in Figure 2. It consists of two parallel executions of the authentication protocol $\lambda_{\mathcal{E}}$ applied to the messages (g^x, g^y) and (g^y, g^x) where A and B act as the sender in one and the receiver in the other. Yet, deniability for SKEME does not immediately follow, from the simulatability of $\lambda_{\mathcal{E}}$. Recall that for a key exchange, we must simulate not only the transcript, but also the output key.

Theorem 3. *If $\mathcal{E}$ is a PA-2 and IND-CPA secure encryption scheme, then SKEME is a concurrently deniable key exchange protocol.*

Proof. We present the idea of the proof. The technical details are similar to those in the proof of Theorem 2.

Assume we are simulating party A (a similar argument holds for B). The simulator SIM chooses x, k_A at random and sends g^x, c_A (as A would do in the real protocol).

When the adversary $\mathcal{M}$ sends g^y, c_B, t_B, the simulator first of all checks if t_B is correct, i.e. equal to $\mathsf{MAC}_{k_A}(g^y, g^x)$. SIM can do this since it knows k_A. If it is not correct, then SIM stops (as A would do in the real protocol).

If t_B is correct, then SIM turns its attention to the ciphertext c_B. In a manner similar to the simulation of $\lambda_{\mathcal{E}}$, if c_B is a "fresh" valid ciphertext, then SIM invokes the extractor $\mathcal{M}^*$ with ciphertext c_B to obtain a plaintext p from which SIM computes a key k. Using this key, SIM computes A's MAC value as $t'_A = \mathsf{MAC}_k(g^x, g^y)$. Since the responses from $\mathcal{M}^*$ are indistinguishable from those of a real decryption oracle then the distribution of MAC values t'_A generated by the simulation is indistinguishable from the distribution of t_A in the real interaction between party A and attacker $\mathcal{M}$.

At this point the simulator has produced a transcript indistinguishable from the real one. However, recall that the complete view includes the key $K = PRF_{k_A}(g^{xy}) \oplus PRF_{k_B}(g^{xy})$. The simulator hence computes, using its knowledge of g^{xy} (SIM knows x) and k_A, the value $K' = PRF_{k_A}(g^{xy}) \oplus PRF_k(g^{xy})$. Again due to the indistinguishability of the keys k and k_B the distribution of the keys K' output by SIM is indistinguishable from the real distribution of keys K.

The simulation in the case of ciphertexts contained in the auxiliary input *aux* is handled identically to the case of the $\lambda_{\mathcal{E}}$ protocol.

SKEME

$\mathsf{Public\ Input}$: $\mathsf{pk}_A, \mathsf{pk}_B$ encryption public keys of A and B respectively.
$\mathsf{Secret\ Input\ of\ A}$: sk_A; $\mathsf{Secret\ Input\ of\ B}$: sk_B

A		B
x, k_A at random $c_A = \mathsf{enc}(\mathsf{pk}_B, k_A)$	$\xrightarrow{g^x\ ,\ c_A}$	
	$\xleftarrow{g^y\ ,\ c_B\ ,\ t_B}$	y, k_B at random and $k'_A = \mathsf{dec}(\mathsf{sk}_B, c_A)$, $t_B = \mathsf{MAC}_{k'_A}(g^y, g^x)$, $c_B = \mathsf{enc}(\mathsf{pk}_A, k_B)$
$t_B \overset{?}{=} \mathsf{MAC}_{k_A}(g^y, g^x)$ and $k'_B = \mathsf{dec}(\mathsf{sk}_A, c_B)$, $t_A = \mathsf{MAC}_{k'_B}(g^x, g^y)$	$\xrightarrow{t_A}$	$t_A \overset{?}{=} \mathsf{MAC}_{k_B}(g^x, g^y)$

Output: Shared key $K = PRF_{k_A}(g^{xy}) \oplus PRF_{k_B}(g^{xy})$

Fig. 2. The three rounds version of SKEME

Notice that Theorem 3 holds for the regular case in which the judge (or distinguisher) is not present during the run of the KE protocol nor it provides inputs to the protocol, but rather is presented with a transcript *a posteriori*. In Appendix C we discuss how SKEME can be made deniable in the case in which the adversary cooperates with the judge before the protocol takes place.

3.3 On Plaintext-Aware Encryptions

In the above discussion we focused on the notion of plaintext-awareness in the standard model (i.e. without random oracle) as introduced in [4]. That work showed that a basic modification of Damgård's scheme [20] is PA-1 secure, while Dent in [21] shows that the Cramer-Shoup scheme [19] is PA-2 secure. Both statements hold under a non-black-box type of assumption known as "Knowledge of exponent assumption" (KEA1) [20, 30, 3], which we recall below.

Let G be a cyclic group of prime order q, and let $g, h \in G$ with $g, h \neq 1$ (and thus both generators of G). The assumption says that for every algorithm $\mathcal{M}$ that on input G, q, g, h outputs (y, z) where $y = g^x$ and $z = h^x$ for some $x \in Z_q$, there exists an algorithm $\mathcal{M}^*$ which outputs x. In other words, the only way for $\mathcal{M}$ to output a pair of elements y, z that have the same discrete log x with respect to two different basis g, h is to know the discrete log x.

Random Oracle Model Schemes. Typical instantiations of SKEME, such as in IKE, use encryption schemes like OAEP which are plaintext-aware in the random oracle model. It is not hard to see that our proof of deniability will also hold for such schemes. Indeed the basic tool to simulate the authentication protocol $\lambda_{\mathcal{E}}$ is a "plaintext extractor" for correct ciphertexts, which is guaranteed to exist by the definition of plaintext-awareness in the random oracle model.

Note that the "programmability feature" of the random oracle is *not* used in the above argument (only the ability to "see" where the adversary queries the oracle.) Thus our simulator is a valid deniability simulator (a simulator that "programs" the random oracle, does not guarantee deniability [42]).

4 Partial Deniability of the SIGMA Protocol

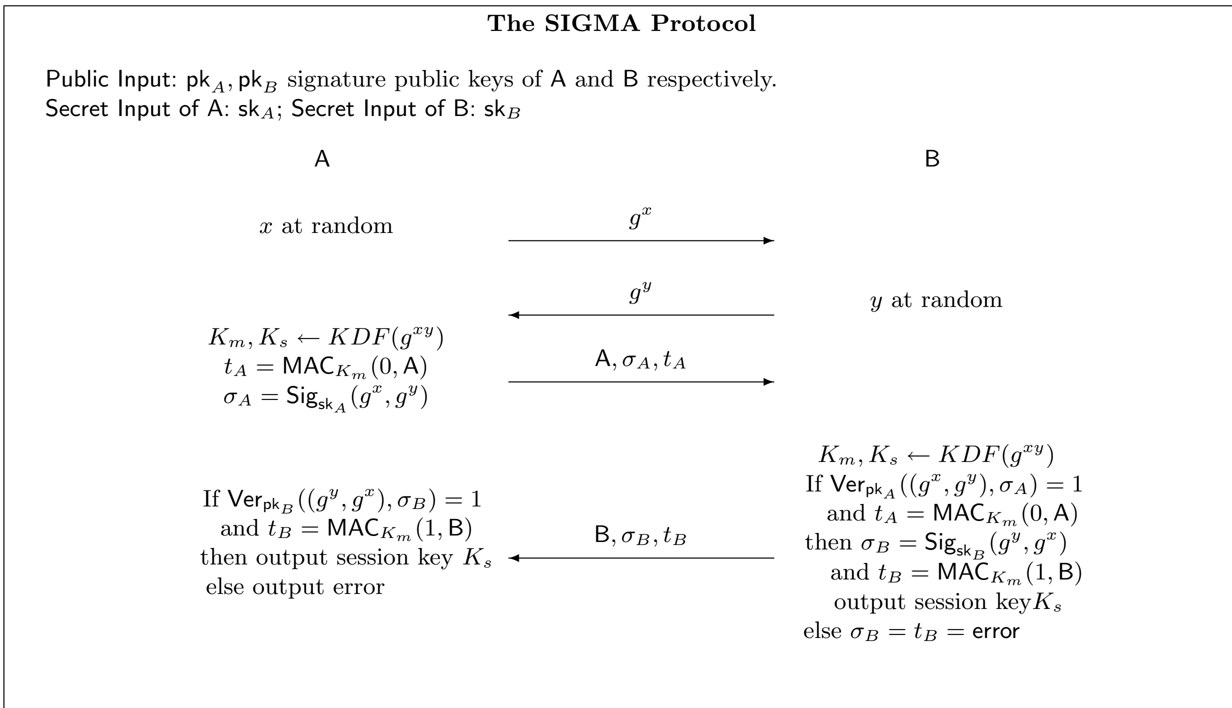
The SIGMA Protocol
Public Input: $\mathsf{pk}_A, \mathsf{pk}_B$ signature public keys of A and B respectively.
Secret Input of A: sk_A; Secret Input of B: sk_B
A
B
x at random
g^x
g^y
y at random
$K_m, K_s \leftarrow KDF(g^{xy})$
$t_A = \mathsf{MAC}_{K_m}(0, \mathsf{A})$
$\sigma_A = \mathsf{Sig}_{\mathsf{sk}_A}(g^x, g^y)$
$\mathsf{A}, \sigma_A, t_A$
$K_m, K_s \leftarrow KDF(g^{xy})$
If $\mathsf{Ver}_{\mathsf{pk}_A}((g^x, g^y), \sigma_A) = 1$
and $t_A = \mathsf{MAC}_{K_m}(0, \mathsf{A})$
then $\sigma_B = \mathsf{Sig}_{\mathsf{sk}_B}(g^y, g^x)$
and $t_B = \mathsf{MAC}_{K_m}(1, \mathsf{B})$
output session keyK_s
else $\sigma_B = t_B = \mathsf{error}$
$\mathsf{B}, \sigma_B, t_B$
If $\mathsf{Ver}_{\mathsf{pk}_B}((g^y, g^x), \sigma_B) = 1$
and $t_B = \mathsf{MAC}_{K_m}(1, \mathsf{B})$
then output session key K_s
else output error

Fig. 3. The four-round version of SIGMA

The SIGMA key-exchange protocol [37] depicted in Figure 3 forms the cryptographic basis for the Internet Key Exchange (IKE) protocol (specifically, the signature-based mode in IKEv1 [31] and the public key mode in IKEv2 [35]). The basic goal of the protocol, proven secure in [12], is to provide a secure Diffie-Hellman exchange authenticated using digital signatures. In addition, the protocol was designed with the privacy goal, referred to as 'identity protection', of hiding the logical identities of the participants from eavesdroppers and active attackers in the network[4]. While deniability was not an explicit design goal of SIGMA (in IKEv1 deniability was offered via the encryption-based mode which uses the SKEME protocol studied in Section 3), we show here that the protocol does provide some significant level of deniability even though it does not achieve the full deniability of SKEME.

Remarks on the protocol description. The 4-round SIGMA protocol is presented in Figure 3. In it each party signs its own DH exponential as well as the peer's exponential and computes a MAC on its own identity. The MAC is computed also over a value 0/1 depending on whether the MAC is computed by the initiator or responder; this serves to prevent reflection attacks. Alternatively, as in the case of IKE, initiator and responder can use different MAC keys (both derived from g^{xy}) in generating their MAC values. The deniability of the protocol is preserved with either technique. On the other hand, we stress that if one adds a differentiator, such as 0/1 to the signatures then the proof of deniability (for the responder) as presented below will not work. Indeed, the proof uses the fact that the signatures produced by the parties do not include evidence of whether the signature was produced in the role of initiator and responder. Therefore, implementations of SIGMA that seek to provide deniability must be careful about modifications to the protocol.

Another aspect of SIGMA that will be of importance in our analysis in Section 4.2 relates to the way the MAC key K_m and session key K_s are derived from the DH value g^{xy}. This key derivation function (KDF) typically consists of three components: (i) computing the DH value g^{xy}, (ii) extracting a key K from g^{xy} via a hashing operation (implemented via SHA-1, a universal hashing scheme, etc.), (iii) using a PRF with key K to derive the two values K_m, K_s (e.g., the first is set to $\mathrm{PRF}_K(0)$ and the latter to $\mathrm{PRF}_K(1)$). We do not specify the way these components are implemented but in the analysis we will need to assume certain properties of these functions. Finally, we stress that when the third and fourth message of SIGMA are encrypted (as needed to provide identity protection) the deniability of the protocol is preserved (the proof is just a straightforward adaptation of the proof presented below).

4.1 Partial Deniability

The challenge in creating a deniable key-exchange protocol that uses digital signatures is that the sole fact that a signature was generated, even if on random inputs, may provide significant information (e.g., that the signer was "alive" or

[4] To achieve identity protection the shown protocol is augmented with an encryption of messages 3 and 4. We omit the encryption part here since it is not necessary for our discussion. We refer to [37] for full details of the protocol.

active). Yet, even in this case the range of deniability may vary widely: from no deniability in the case of a protocol that signs the peer's identity (as the ISO protocol discussed in the introduction) to a protocol that only signs self-generated random information. Here we investigate the position of SIGMA in this "deniability range". We'll see that the provision of not signing the peer's identity (but rather MACing one's own identity), needed to achieve identity protection, also provides the basis for deniability.

Let's consider first the role of the initiator, or A. The core observation is that A will sign g^y irrespective of who generated it. For example, consider an attacker $\mathcal{M}$ that encodes in the exponent y (used to generate the value g^y sent to A) its own identity (e.g., choosing y to be a signature by this attacker). The fact that A will sign g^y says nothing about whether A talked or not to $\mathcal{M}$ (or even if she was willing to talk to $\mathcal{M}$) since A will sign g^y regardless of who sent it. In other words, A's transcripts are *peer-independent*. The case of the responder B is similar to the above and its transcript is also "peer-independent". However, we will see later that the fact that B sends its signature on (g^y, g^x) *after* verifying the sender's identity may provide some extra information under special circumstances. This peer-independence property provides a limited, yet meaningful and significant form of deniability.

To formalize it we resort (again) to the definitional approach of Dwork, Naor and Sahai [27] who introduced a weakened notion of deniability, in the context of authentication protocols, to deal with situations similar to the above (e.g., when the authentication algorithm uses digital signatures). This relaxed definition from [27] still follows the simulation paradigm but the simulator is allowed to interact with an oracle representing the real sender (or prover) on messages other than the one provided as input to the simulator. This captures the fact that whatever is learned from the authentication protocol is *independent* of the authenticated message. This relaxation is useful in proving the limited deniability of certain protocols, yet its exact "practical meaning" needs to be assessed in a per-application basis. For example, such a definition does not preclude the possibility to deduce (and prove to a third party) the number of messages a party has authenticated; this may be a significant information in some cases (in the extreme, a protocol may be built to authenticate a single message, e.g., "attack", and in this case the above form of partial deniability from [27] is too weak – similar caveats will apply to our partial deniability notion for KE protocols defined below; see also Section 4.3).

Roughly speaking, we will say that a key-exchange protocol is *partially deniable for the initiator* if the runs of the (honest) initiator A with a given responder B are indistinguishable from runs with any other responder B'. In the formal definition we provide the initiator's simulator SIM_I, simulating an initiator A and acting on input $(\mathsf{B}, \mathsf{pk}_B)$, with one of two oracles: an oracle that acts as the real initiator A running with peer B' where $\mathsf{B}' \neq \mathsf{B}$, or an oracle that acts as A running as the responder with peer B'. Similarly, we define partial deniability for the responder. Then we will say that the protocol is *partially deniable* if it is deniable for the initiator and responder. This is formalized in Definition 3 below.

Notation. Let Σ be a key-exchange protocol with interactive machines Σ_I, Σ_R defining the roles of the (honest) initiator and responder, respectively. We use the symbol $\Sigma_I^C(D, \mathsf{pk}_D)$ to denote the interactive machine implementing a honest party C as the initiator in a run of protocol Σ with peer D and peer's public key pk_D. Implicit in this notation is that Σ_I^C has as input the secret and public keys of C. The responder machine $\Sigma_R^C(D, \mathsf{pk}_D)$ is defined analogously.
Formal note: in some KE protocols, including SIGMA, parties can be initially activated without the values of the peer's identity and/or public key; hence the pair (D, pk_D) is not necessarily provided as input at the onset of the protocol but only when communicated by the peer (in the case of SIGMA this happens in the third and fourth messages).

Definition 3. *Let $\Sigma = (\Sigma_I, \Sigma_R)$ be a key-exchange protocol. We say that Σ_I is* partially deniable with respect to an I-oracle (resp. R-oracle) *if for any adversary $\mathcal{M}$ and any (honest) party C, the interaction between C as initiator with $\mathcal{M}$ as responder can be simulated (as in Definition 2) by a simulator SIM_I that is given oracle access to $\Sigma_I^C(D, \mathsf{pk}_D)$ (resp. $\Sigma_R^C(D, \mathsf{pk}_D)$) where pk_D is a public key chosen independently of $\mathcal{M}$'s public key $\mathsf{pk}_{\mathcal{M}}$.*
We say that Σ_I is partially deniable *if it is partially deniable with respect to an I-oracle or with respect to an R-oracle. The definition of Σ_R being partially deniable is similar.*
Finally, we say that $\Sigma = (\Sigma_I, \Sigma_R)$ is partially deniable *if both Σ_I and Σ_R are partially deniable.*

Concurrency. For ease of presentation the above definition is formulated in terms of a single "stand-alone" execution of the protocol. Adding full concurrency to the definition is straightforward (see Definition 2). More importantly, we note that the proof of partial deniability of SIGMA (Theorem 4) avoids rewinding, and thus holds in the concurrent model.

Auxiliary input. The treatment of auxiliary input, omitted in the above simplified definition, is identical to the case of Definition 2; in particular, the proof of Theorem 4 below holds with respect to such auxiliary input. What is important to stress is that the SIGMA protocol has the salient property that it is (partially) deniable even if the judge provides DH values to the attacker to be used in the protocol and for which the attacker does not know the exponent (see the discussion on off-line/on-line judges in Section 2). Indeed, when $\mathcal{M}$ acts as an initiator then receiving g^x (but not x) from a third party does not allow $\mathcal{M}$ to produce the correct third protocol message and hence the execution is aborted without the responder generating the authentication information in message 4. In the case that $\mathcal{M}$ acts as responder then a third-party provided g^y makes no difference since the initiator authenticates g^y independently of the peer (and without "inspecting" the g^y value itself).

The meaning of partial deniability. In Section 4.3 we further discuss the "real-life" semantics and relevance of our notion of partial deniability.

4.2 Deniability Analysis of SIGMA

Here we show that SIGMA as depicted in Figure 3 is partially deniable as in Definition 3. The main difficulties in the simulation of the protocol are (i) the generation of the signatures on behalf of the simulated parties, and (ii) the computation by the simulator of the session key K_s and the MAC key K_m. Note that producing K_s is a necessary condition for satisfying the definition of deniability and partial deniability (Definitions 2 and 3) while computing (or learning) K_m is a necessary condition to simulate the generation and verification of the MAC values exchanged in the protocol.

For point (i) the simulator will use the real parties as oracles (as allowed by the definition of partial deniability) to produce the signatures. Specifically, the simulation of a given party C acting as initiator will use an oracle to the real party C acting as initiator. The simulation of C as responder will also use an oracle to the real party C acting as initiator (rather than as a responder – see discussion in Section 4.3).

For point (ii), since each of the exponentials g^x, g^y are chosen by either the simulator's oracles or by the adversary then the corresponding exponents x, y are not known to the simulator who thus cannot easily compute the values g^{xy}, K_m or K_s. We solve this problem as follows. If the attacker itself cannot compute its own MAC values then the simulator will be able to succeed without computing or verifying these MAC values (in this case, the simulated party would abort before having to compute the MAC values). However, if the attacker can compute the MAC then, intuitively, the simulator (which has non-black-box access to the attacker) can learn these values as well.

However, what we really need is for the simulator to learn not just the MAC values but the keys K_m and K_s. In most natural/practical implementations of the key derivation function of SIGMA one will have the property that for the adversary to compute the right MAC values, it has to compute the right MAC key and to do so it has to compute the PRF key from which also K_s is computed. This, however, is not a necessary condition that follows just from the regular definitions of MAC and PRF: one may be able to construct artificial functions where the attacker succeeds in computing its own correct MAC value without necessarily having to compute the key K_m (note that the key K_m does not have to be random as the attacker can influence it via the choice of the DH exponential). Moreover, one can envision a situation where the attacker can prove that it could have not possibly known how to compute the MAC produced by its peer to an exchange in which case the exchange (and possibly the following communication) may not be deniable. Therefore, our proof of SIGMA will apply to KDF construction where the above artificial situation does not arise. We formalize this via the following "key awareness" assumption.

The Key-Awareness Assumption for SIGMA's KDF. *We say that a KDF procedure for SIGMA has the* Key-Awareness property *if for all deniability attackers $\mathcal{M}$ against the protocol the following holds. If on exchange of DH values X, Y an attacker $\mathcal{M}$ computes a correct value* $\mathsf{MAC}_{K_m}(t)$*, for some input t, where K_m*

is the MAC key derived by the KDF from X, Y, then there is an "extractor machine" $\mathcal{M}'$ that on the same inputs of $\mathcal{M}$ outputs the keys K_m and K_s.[5]

In other words, there are no "shortcuts" available for the attacker in computing the MAC without going through the key derivation steps and explicitly computing the MAC key K_m as well as the companion key K_s. We note that the above very informal definition of the Key-Awareness can be formalized in ways similar to other non-black box "extraction assumptions" such as the "knowledge of exponent" and plaintext awareness assumptions discussed in Section 3. We omit the gory details here.

As we said earlier we expect most natural implementations of key derivation in SIGMA to have the above property, in particular when the hashing, PRF and MAC are implemented using HMAC or CBC-MAC using strong hash and block-cipher functions as in the implementation of SIGMA in IKE. Of course, we cannot prove this, but we have to explicitly assume it.

Next we show that this assumption holds when the hash function H used to derive the PRF key $K = H(g^{xy})$ is modelled as a random oracle; in this case, the whole key derivation has the key awareness property. Namely, we show that the simulator can extract the correct keys K_m and K_s from the run of the attacker, provided the latter computes correct MAC values. (Interestingly, while intutive the following argument contains some unexpected subtleties.)

Due to the randomness of H, if the attacker $\mathcal{M}$ does not compute g^{xy} then the key $K = H(g^{xy})$ (and thus K_m) is indistinguishable from random for $\mathcal{M}$. In this case, the attacker cannot possibly compute the correct value of a MAC under K_m. Thus, the simulator (which has access to the oracle H) can check the inputs to H provided by $\mathcal{M}$ and hence learn K, and with it both K_m and K_s. There is however a problem in this argument. The simulator will not be able to compute, in general, the value g^{xy} by itself; so how will it know which of the inputs to H was the real value g^{xy}? (In particular, $\mathcal{M}$ may use a key K, or K_m, derived from an output of H on a point different than g^{xy}). The solution to this problem uses the fact that in the simulation the simulator SIM will have an example of a *correct* value of a MAC, computed under the *correct* key K_m, produced by a real player! Thus, in this case, SIM proceeds as follows. It considers each output of H as a candidate PRF key K^* and derives from it two candidate keys K_s^* and K_m^*. Now, SIM, uses the candidate key K_m^* to verify the MAC value received from the real player. If the verification succeeds then SIM assumes K_m^* to be the correct MAC key K_m (and K_s^* to be the correct session key). It is easy to see that a wrong candidate key K_m^* will succeed in verifying the real MAC with negligible probability. Indeed, since the distribution of wrong K_m^* keys is uniform (and independent from K_m) then the latter probability is at most as the probability to forge a MAC and hence negligible. (Clearly, if two independent random keys have a high probability of producing, or verifying,

[5] In the case in which the protocol uses additional keys, such as directional MAC keys, encryption keys, etc, then we assume that the extractor returns all these keys; alternatively, if all these keys are derived from a single PRF key K then it suffices that extractor returns this key K.

the same MAC value then one can forge MAC values computed under a secret random key by simply re-computing the MAC value under another random and independent key.)

Another subtlety in the above argument is what is meant by the "right value" of g^{xy}. What happens if the attacker chooses, say, a value Y (instead of g^y) not in the subgroup generated by g (as in the Lim-Lee attacks [39])? In this case the value g^{xy} is not even well-defined. However, since in this case we will have that the value $X = g^x$ was chosen by a real (and honest) player then the value Y^x is well defined and it is this value that we actually refer to as the "right g^{xy}" (conversely, when X is the value chosen by the attacker and $Y = g^y$ is chosen by the honest player then the "real value of g^{xy}" refers to X^y).

NOTE. We note that the above simulation requires no rewinding and hence it does not break the concurrency of the deniability property. Also worth noting is that as pointed out by Pass [42] one has to be careful when arguing deniability using the random oracle model. Specifically, one cannot use in such an argument the so called "programmability feature" of random oracles. We note that this property is *not* used in the above argument. (We only use the ability to "see" where the adversary queries the oracle.)

Summarizing, we prove the following Theorem.

Theorem 4. *The SIGMA protocol from Figure 3 with a key-aware key derivation is partially deniable according to Definition 3. (In particular, this is the case if one models the hashing of g^{xy} in SIGMA as a random oracle.)*

Proof. We first show that Σ_I is partially deniable with respect to an I-oracle. That is, we show a simulator SIM_I that simulates the interaction between the adversary $\mathcal{M}$ as responder with a (honest) initiator C where SIM_I is given oracle access to $\Sigma_I^C(D, \mathsf{pk}_D)$ and pk_D is independent from $\mathsf{pk}_{\mathcal{M}}$. In this case SIM_I is very simple, it uses the oracle $\Sigma_I^C(D, \mathsf{pk}_D)$ to produce the messages on behalf of $\Sigma_I^C(\mathcal{M}, \mathsf{pk}_{\mathcal{M}})$ and passes these messages to and from $\mathcal{M}$. Specifically, SIM_I activates $\Sigma_I^C(D, \mathsf{pk}_D)$ and gets g^x which it passes to $\mathcal{M}$; it then passes the response g^y from $\mathcal{M}$ back to $\Sigma_I^C(D, \mathsf{pk}_D)$. When the latter produces its second message with the triple $C, \sigma_C = \mathsf{Sig}_{\mathsf{sk}_C}(g^x, g^y), t_C = \mathsf{MAC}_{K_m}(0, C)$, the simulator SIM_I passes it to $\mathcal{M}$. So far the simulation of C is perfect (it is actually produced by C itself and it does not depend on the peer D – actually, in many applications of SIGMA, C does not necessarily know at this point who the peer is). If $\mathcal{M}$ returns its final message with the triple $\mathcal{M}, \sigma_{\mathcal{M}} = \mathsf{Sig}_{\mathsf{sk}_{\mathcal{M}}}(g^x, g^y), t_{\mathcal{M}} = \mathsf{MAC}_{K_m}(1, \mathcal{M})$, the simulator needs to check the validity of the signature (this is easy as SIM_I knows $\mathcal{M}$'s public key) as well as the validity of the MAC value $t_{\mathcal{M}}$. For this we use the key-aware KDF assumption by which either $t_{\mathcal{M}}$ is the wrong MAC value or else SIM_I extracts the correct keys K_m and K_s. If the extractor does not return a MAC key K_m then SIM_I (on behalf of $\Sigma_I^C(\mathcal{M}, \mathsf{pk}_{\mathcal{M}})$) aborts the run of the protocol (in this case the MAC returned by $\mathcal{M}$ was wrong), and outputs the simulated view. Note that the key K_s is not output by SIM_I since in the real protocol C does not produce a session key in this case. If the correct K_m, K_s are recovered then SIM_I uses K_m to validate $t_{\mathcal{M}}$. If correct, it completes its run by outputting the simulated view together with the value of the session key K_s. It is easy to see that the simulation is perfect (up to a negligible probability of error in the KDF extractor).

Next, we show that Σ_R is partially deniable with respect to an I-oracle. That is, we show a simulator SIM_R that simulates the interaction between the adversary $\mathcal{M}$ as initiator with a (honest) responder C where SIM_R is given oracle access to $\Sigma_I^C(D, \mathsf{pk}_D)$ and pk_D is independent from $\mathsf{pk}_{\mathcal{M}}$. This simulation is somewhat more involved than the previous one due to the differences of roles of C as responder in the simulation and as initiator under the oracle $\Sigma_I^C(D, \mathsf{pk}_D)$. Specifically, when $\mathcal{M}$ sends its initial message g^x, the simulator SIM_R activates the oracle $\Sigma_I^C(D, \mathsf{pk}_D)$ and gets an initial value $g^{x'}$. Now, SIM_R sets g^y to the value $g^{x'}$ and sends g^y back to $\mathcal{M}$. It also sets $g^{y'}$ to the value of g^x and sends $g^{y'}$ to $\Sigma_I^C(D, \mathsf{pk}_D)$ who responds with $C, \sigma_C = \mathsf{Sig}_{\mathsf{sk}_C}(g^{x'}, g^{y'}), t_C = \mathsf{MAC}_{K_m}(0, C)$ where K_m is derived via $KDF(g^{x'y'})$ which is equivalent to $KDF(g^{xy})$ due to the equality $g^x = g^{x'}$ and $g^y = g^{y'}$. When $\mathcal{M}$ responds with its third message $\mathcal{M}, \sigma_{\mathcal{M}} = \mathsf{Sig}_{\mathsf{sk}_{\mathcal{M}}}(g^x, g^y), t_{\mathcal{M}} = \mathsf{MAC}_{K_m}(0, \mathcal{M})$, then SIM_R proceeds as follows. If verification of $\mathcal{M}$'s signature fails then the protocol aborts. Else, SIM_R applies an extractor to get K_m, K_s. If the keys are not returned by the extractor SIM_R aborts the protocol since the MAC $t_{\mathcal{M}}$ must be wrong. Otherwise, SIM_R uses the extracted K_m to calculate the MAC value $t'_C = \mathsf{MAC}_{K_m}(1, C)$ (note that the value t_C output by $\Sigma_I^C(D, \mathsf{pk}_D)$ used a '0' in the mac instead of '1'), and sends to $\mathcal{M}$ the final message C, σ_C, t'_C which corresponds exactly to the message C would have sent to $\mathcal{M}$ if C acted as responder (in particular note that $\sigma_C = \mathsf{Sig}_{\mathsf{sk}_C}(g^{x'}, g^{y'}) = \mathsf{Sig}_{\mathsf{sk}_C}(g^x, g^y)$). The simulator completes its run by outputting the full view including the session key K_s.

4.3 Discussion on Partial Deniability

It is important to understand the "real-life" semantics of our notion of partial deniability. The idea behind our definition is that the transcript available to the adversary $\mathcal{M}$ could have been produced by a party C when interacting with any other party D (this is what we refer to as the "peer independence" property). Thus C can deny having been involved with $\mathcal{M}$. A point to notice is the role of C during this "claimed" interaction with D. The simpler case is when C acts as initiator: in this case the information generated by C is totally independent of the peer's identity and hence interaction with any specific peer can be denied by the initiator. In the case that C runs as responder, the "peer independence" property can be formally proven provided that C also runs as initiator in *other* instances of the protocol. In other words, the authentication information created by C as responder is simulatable from the information created by C as initiator. How common is it in real-life protocols that parties run as both initiators and responders? Clearly, this depends on the application. For example, in applications of SIGMA to Instant Messaging (as in [7, 24]) the common case is that end-points to the IM service act as both responders and initiators. In contrast, in a typical client-server configuration of IPSec, parties will run as either initiators or responders but not both (fortunately, in these cases clients usually run as initiators so they are fully protected by deniability). However, as we see next, SIGMA provides some level of deniability also in cases where parties act exclusively as responders and not as initiators.

Deniability for parties who only act as responders. For a party, B, that only acts as a responder (never as an initiator), SIGMA still provides some form of deniability whose significance may depend on the application. Indeed, note that there is nothing explicit in the authentication information sent by the responder that ties it to a specific peer. The problem, however, is that B will send this information only after verifying the identity of the peer. Let's consider an example. Say, Charlie, acting as initiator, sends Bob g^x where x encodes Charlie's signature on some value. Bob signs this g^x and later Charlie brings to court Bob's signature and the value x showing that this value was generated by him (Charlie). In itself this proves nothing about Bob having *knowingly* communicated with Charlie: g^x could have been sent by David, a party with whom Bob was willing to talk. In other words, David could have been collaborating with Charlie in "framing" Bob (or maybe Charlie just broke into David's computer). In this sense, Bob enjoys deniability even though it acts as responder-only in SIGMA. On the other hand, one can imagine cases where additional "circumstantial evidence" may make it harder for Bob to deny, especially since Bob needs to convince that Charlie was using someone with whom Bob was willing to communicate for mounting the attack. This may be difficult to do, for example, in the following situation: Charlie is a malicious web site that decides to disclose identities of its visitors/customers (who would prefer to remain anonymous) by sending each customer a g^x value that "ties" the communication to this web site as above. Now, if Bob is one of these customers it will have to convince the judge that someone with whom Bob is willing to talk was colluding with (or controlled by) the malicious website.

The above considerations apply also to the case of an implementation of SIGMA that adds information under the signatures that make the signatures produced by C as initiator distinguishable from those generated by C as responder (the addition of such information is not needed for the security of SIGMA and is certainly not recommended in any setting where deniability is significant).

Deniability of SIGMA-I. We remark that the 3-message variant of SIGMA, called SIGMA-I [37], is partially deniable, according to Definition 3, *only for the responder.* In this case it is the responder's behavior which is peer-independent since his signature is produced before seeing the identity of the initiator. The initiator, Alice, in SIGMA-I is in a position similar to the responder in SIGMA-R (Fig. 3) since she signs after seeing the identity of the other peer. Unlike the SIGMA-R case, however, the initiator Alice may not be able to claim that her signature on (g^x, g^y) was performed in her role of responder. Indeed, a malicious responder could choose his DH value dependent on Alice's DH value, and this is never the case when Alice acts as responder. In other words, the signatures of initiators and responders can be made distinguishable by a dishonest peer in SIGMA-I, something that is not possible in SIGMA-R. Thus SIGMA-R may be somewhat preferable in the deniability setting.

FInal note. The deniability limitations of SIGMA follow from the use of signatures (essential to the protocol) and the fact that these signatures are applied to a peer-provided value. The latter issue can be avoided if one replaces the peer's DH value under the signature with a freshness value not chosen by the peer, such as a non-repeating counter or a timestamp; however, these values are seldom available, or secure enough, in practical settings. As said earlier, a more essential limitation of partial deniability is that having a signature of a party, even on random information, is sufficient proof to show that the party was "alive". Moreover, it is easy to see that in the case of SIGMA an attacker can encode into its DH value, information that will allow to prove not only that a party was alive but that it was alive after certain time or event (for example, the attacker can encode into the DH value the hashing of today's New York Times). Leaving a proof of such information seems unavoidable in any protocol that needs to include some "freshness guarantee" inside a signature to prevent its replay.

References

1. M. Bellare, R. Canetti and H. Krawczyk, A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols, *proc. of* 30th *Symposium on Theory of Computing (STOC)*, ACM, pp. 419–428, 1998.
2. M. Bellare, A. Desai, D. Pointcheval and P. Rogaway, Relations among Notions of Security for Public-Key Encryption Schemes, *Advances in Cryptology – proc. of CRYPTO '98, LNCS 1462*, Springer-Verlag, pp. 26–45, 1998.
3. M. Bellare and A. Palacio, The Knowledge-of-Exponent Assumptions and 3-Round Zero-Knowledge Protocols, *Advances in Cryptology – proc. of CRYPTO '04, LNCS 3152*, Springer-Verlag, pp. 273–289, 2004.
4. M. Bellare and A. Palacio, Towards Plaintext-Aware Public-Key Encryption without Random Oracles, *Advances in Cryptology – proc. of ASIACRYPT '04, LNCS 3329*, Springer-Verlag, pp. 48–62, 2004.
5. M. Bellare and P. Rogaway, Entity authentication and key distribution, *Advances in Cryptology, – proc. of CRYPTO '93, LNCS 773*, Springer-Verlag, pp. 232–249, 1994.
6. M. Bellare and P. Rogaway, Optimal Asymmetric Encryption, *Advances in Cryptology – proc. of EUROCRYPT '94, LNCS 950*, Springer-Verlag, pp. 92–111, 1994.
7. N. Borisov, I. Goldberg and E. Brewer, Off-the-Record Communication, or, Why Not To Use PGP, *proc. of the 2004 ACM Workshop on Privacy in the Electronic Society*, ACM Press, pp. 77–84, October 2004.
8. C. Boyd, W. Mao and K. Paterson, Key Agreement using Statically Keyed Authenticators, *proc. of Applied Cryptography and Network Security: Second International Conference (ACNS 2004), LNCS 3089*, Springer-Verlag, pp. 248–262, 2004.
9. R. Canetti, C. Dwork, M. Naor and R. Ostrovsky. *Deniable Encryption.* Advances in Cryptology – proc. of CRYPTO '97, lNCS 1294, pp.90-104, Springer, 1997.
10. R. Canetti and H. Krawczyk, Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels, *Advances in Cryptology – proc. of EUROCRYPT '01, LNCS 2045*, Springer-Verlag, pp. 453–474, 2001.

11. R. Canetti and H. Krawczyk, Universally Composable Notions of Key Exchange and Secure Channels, *Advances in Cryptology – proc. of EUROCRYPT '02, LNCS 2332*, Springer-Verlag, pp. 337–351, 2002. Full version available at `http://eprint.iacr.org/2002/059.`
12. R. Canetti and H. Krawczyk, Security Analysis of IKE's Signature-based Key-Exchange Protocol, *Advances in Cryptology – proc. of CRYPTO '02, LNCS 2442*, Springer-Verlag, pp. 143–161, 2002.
13. D. Chaum, Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms, *Communications of the ACM*, vol. 24 n. 2, February 1981.
14. D. Chaum, Blind Signatures for Untraceable Payments, *Advances in Cryptology – proc. of CRYPTO '82*, D. Chaum, R.L. Rivest, & A.T. Sherman (Eds.), Plenum, pp. 199–203, 1982.
15. D. Chaum, Security Without Identification: Transaction Systems to Make Big Brother Obsolete, *Communications of the ACM*, vol. 28 n. 10, pp. 1030–1044, October 1985.
16. D. Chaum and H. van Antwerpen, Undeniable Signatures, *Advances in Cryptology – proc. of CRYPTO '89, LNCS 435* Springer-Verlag, pp. 212–226, 1990.
17. S. Chawla, C. Dwork, F. McSherry, A. Smith and H. Wee, Toward Privacy in Public Databases, *proc. of* 2nd *Theory of Cryptography Conference (TCC 2005)*, p. 363–385, 2005.
18. B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan, Private Information Retrieval, *proc. of* 36th *FOCS*, pp. 41–50, 1995
19. R. Cramer and V. Shoup, A Practical Public-Key Cryptosystem Secure Against Adaptive Chosen Ciphertexts Attacks, *Advances in Cryptology – proc. of CRYPTO '98, LNCS 1462*, Springer-Verlag, pp. 13–25, 1998.
20. I. Damgard, Towards Practical Public Key Systems Secure Against Chosen Ciphertext Attacks, *Advances in Cryptology – proc. of CRYPTO '91, LNCS 576*, Springer-Verlag, pp. 445–456, 1992.
21. A. Dent, Cramer-Shoup is Plaintext-Aware in the Standard Model, *Advances in Cryptology – proc. of EUROCRYPT '06, LNCS 4004*, Springer-Verlag, pp. 289–307, 2006
22. W. Diffie and M.E. Hellman, New Directions in Cryptography, *IEEE Transactions on Information Theory*, vol. 22 n. 6, pp. 644–654, 1976.
23. M. Di Raimondo and R. Gennaro, New Approaches for Deniable Authentication, *proc. of* 12nd *ACM Conference on Computer and Communications Security (CCS '05)*, ACM Press, pp. 112–121, 2005.
24. M. Di Raimondo, R. Gennaro and H. Krawczyk, Secure Off-the-Record Messaging, *proc. of* 2nd *ACM Workshop of Privacy in the Electronic Society (WPES'05)*, ACM Press, pp. 81–89, 2005.
25. D. Dolev, C. Dwork and M. Naor, Non-Malleable Cryptography, *SIAM Journal on Computing*, vol. 30 n. 2, pp. 391–437, April 2000.
26. C. Dwork and K. Nissim, Privacy-Preserving Datamining on Vertically Partitioned Databases, *Advances in Cryptology – proc. of CRYPTO '04, LNCS 3152*, Springer-Verlag, pp. 528–544, 2004.
27. C. Dwork, M. Naor and A. Sahai, Concurrent Zero-Knowledge, *proc. of* 30th *Symposium on Theory of Computing (STOC)*, ACM Press, pp. 409–418, 1998. Full version on `http://www.wisdom.weizmann.ac.il/~naor/onpub.html`.
28. W. Diffie, P. Van Oorschot and M. Wiener, Authentication and Authenticate Key Exchange, *Designs, Codes and Cryptography*, n. 2, pp. 107–125, 1992.

29. S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof-systems, *SIAM Journal on Computing*, vol. 18 n. 1, pp. 186–208, February 1989.
30. S. Hada and T. Tanaka, On the Existence of 3-Round Zero-Knowledge Protocols, *Advances in Cryptology – proc. of CRYPTO '98, LNCS 1462*, Springer-Verlag, pp. 408–423, 1998.
31. D. Harkins and D. Carrel, eds. The Internet Key Exchange (IKE). *RFC 2409*, November 1998.
32. ISO/IEC IS 9798-3, "Entity authentication mechanisms — Part 3: Entity authentication using asymmetric techniques", 1993.
33. M. Jakobsson, K. Sako and R. Impagliazzo, Designated Verifier Proofs and Their Applications, *Advances in Cryptology – proc. of EUROCRYPT '96, LNCS 1070*, Springer-Verlag, pp. 143–154, 1996.
34. J. Katz, Efficient and Non-Malleable Proofs of Plaintext Knowledge and Applications, *Advances in Cryptology – proc. of EUROCRYPT '03, LNCS 2656*, Springer-Verlag, pp. 211–228, 2003.
35. C. Kaufman, ed., Internet Key Exchange (IKEv2) Protocol, draft-ietf-ipsec-ikev2-17.txt, September 2004 (pending RFC).
36. H. Krawczyk, SKEME: a versatile secure key exchange mechanism for Internet, *proc. of 1996 IEEE Symposium on Network and Distributed System Security (SNDSS '96)*, pp. 114–127.
37. H. Krawczyk, SIGMA: The 'SiGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols, *Advances in Cryptology – proc. of CRYPTO '03, LNCS 2729*, Springer-Verlag, pp. 400–425, 2003. Available at `http://www.research.ibm.com/security/sigma.ps`
38. Y. Lindell and B. Pinkas, Privacy Preserving Data Mining, *Journal of Cryptology*, vol. 15 n. 3, pp. 177–206, Springer, 2002.
39. C.H. Lim and P.J. Lee, A Key Recovery Attack on Discrete Log-based Schemes Using a Prime Order Subgroup, *Advances in Cryptology – proc. of CRYPTO '97, LNCS 1294*, Springer-Verlag, pp. 249–263, 1997.
40. W. Mao and K.G. Paterson, On the Plausible Deniability Feature of Internet Protocols, *Manuscript.*
41. M. Naor, Deniable Ring Authentication, *Advances in Cryptology – proc. of CRYPTO '02, LNCS 2442*, Springer-Verlag, pp. 481–498, 2002.
42. R. Pass, On Deniability in the Common Reference String and Random Oracle Model, *Advances in Cryptology – proc. of CRYPTO '03, LNCS 2729*, Springer-Verlag, pp. 316–337, 2003.
43. C. Rackoff and D.R. Simon, Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack, *Advances in Cryptology – proc. of CRYPTO '91, LNCS 576*, Springer-Verlag, pp. 433–444, 1992.
44. R. Rivest, A. Shamir and Y. Tauman, How to Leak a Secret, *Advances in Cryptology – proc. of ASIACRYPT '01, LNCS 2248*, Springer-Verlag, pp. 552–565, 2001.
45. V. Shoup, On Formal Models for Secure Key Exchange, IBM Research Report RZ 3120, April 1999.

A Definitions

A.1 Authentication Protocols

Here we recall the notion of an *authentication protocol* (in the public-key model) as defined by Dwork, Naor and Sahai [27].

An *authentication* protocol consists of a triple $(\mathsf{AKG},\mathsf{S},\mathsf{R})$: where AKG is the *key generation* algorithm which on input 1^n, where n is the security parameter, outputs a public/secret key pair $(\mathsf{sk},\mathsf{pk})$; S and R are interactive Turing Machines called the *sender* and *receiver*, respectively. They run on common input pk and a message m, while S also holds the secret key sk. At the end R either accepts or rejects.

We consider a *concurrent* communication model [43, 27, 2] in which executions of the protocol can be arbitrarily scheduled and interleaved by the adversary. We stress that this requirement is fundamental to claim security for protocols ran in practical scenarios and open networks like the Internet.

Informally, a *secure authentication* protocol must be *complete* (S should always make R accept) and *sound* (or *unforgeable*): no forger $\mathcal{F}$ interacting with R on common input (pk, m) (where pk is the public key of S) should make R accept, even if he has oracle access to S authenticating messages $m' \neq m$. More formally:

- AKG is the *key generation* algorithm: on input 1^n, where n is the security parameter, it outputs a public/secret key pair $(\mathsf{sk},\mathsf{pk})$;
- S and R, called the *sender* and *receiver*, respectively, are interactive Turing machines; S runs on input a key pair $(\mathsf{sk},\mathsf{pk})$ and a message m; while R runs only on pk and m. With $(\mathsf{S}(\mathsf{sk}), \mathsf{R})(\mathsf{pk}, m)$ we denote the output of this interaction (by convention, this output is a single bit representing a successful/failed authentication).

$(\mathsf{AKG},\mathsf{S},\mathsf{R})$ is called a *secure authentication* protocol if the following properties hold:

Completeness. For any message m and any key pair $(\mathsf{sk},\mathsf{pk})$, $(\mathsf{S}(\mathsf{sk}), \mathsf{R})(\mathsf{pk}, m) = 1$.

Soundness. For any forger $\mathcal{F}$ with oracle access to $\mathsf{S}(\mathsf{sk}, \mathsf{pk}, \cdot)$:

$$Pr[(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{AKG}(1^n)\ ;\ (\mathcal{F}^{\mathsf{S}(\mathsf{sk},\mathsf{pk},\cdot)}, \mathsf{R})(\mathsf{pk}, m) = 1] \leq \mathsf{negl}(n)$$

where m was not queried to the oracle S by $\mathcal{F}$. (That is, the probability that $\mathcal{F}$, pretending to be S, successfully authenticates a new message m to R is negligible even if $\mathcal{F}$ is allowed to interact with S on inputs other than m.)

Notice that the above definition intrinsically considers a concurrent scenario since $\mathcal{F}$ can arbitrarily schedule and interleave its interactions with the real sender S, while trying to fool the real receiver R.

A.2 Encryption Notions

An encryption scheme $\mathcal{E}$ is a triple $(\mathsf{gen},\mathsf{enc},\mathsf{dec})$ where

- gen is the key generation algorithm. On input 1^k, where k is the security parameter, it outputs a public/secret key pair $(\mathsf{sk},\mathsf{pk})$.

– enc is the encryption algorithm. On input a message m and a the public key pk it outputs a ciphertext c.
– dec is the decryption algorithm. On input a ciphertext c and the secret key sk it outputs either $\perp$ or a message m.

The obvious requirement is that if $(\mathsf{sk}, \mathsf{pk}) = \mathsf{gen}(1^k)$ and $c = \mathsf{enc}(\mathsf{pk}, m)$ then $m = \mathsf{dec}(\mathsf{sk}, c)$.

We say that $\mathcal{E}$ is semantically secure (or IND-CPA) if for any two messages m_1, m_2 and for pk chosen according to gen, the two distributions c_1, c_2, where $c_i = \{\mathsf{enc}(\mathsf{pk}, m_i)\}$, are indistinguishable.

Consider now the following game: choose sk,pk according to $\mathsf{gen}(1^n)$. Then give the adversary oracle access to $\mathsf{dec}(\mathsf{sk}, \cdot)$ i.e. allow $\mathcal{M}$ to get decryptions of ciphertexts of her choice. At one point $\mathcal{M}$ outputs two messages m_0, m_1: choose a random bit b and compute $c = \mathsf{enc}(\mathsf{pk}, m_b)$ and return it to $\mathcal{M}$. Now restrict $\mathcal{M}$ access to $\mathsf{dec}(\mathsf{sk}, \cdot)$ to any ciphertext except c until $\mathcal{M}$ outputs a bit b'. Denote with $adv_{\mathcal{M}}(n) = Pr[b = b'] - 1/2$. We say that $\mathcal{E}$ is semantically secure against adaptive chosen ciphertext attack (or IND-CCA2, or CCA2-secure) [25] if for all adversary $\mathcal{M}$, $adv_{\mathcal{M}}(n)$ is negligible in n.

A.3 Definition of Plaintext-Awareness

In this section we recall the notion of plaintext-awareness (PA) for an encryption scheme. Intuitively, an encryption scheme is Plaintext-Aware (PA) if the "only" way, that an adversary can produce a valid ciphertext, is to apply the encryption algorithm on a given message. In other words, any adversary against a PA scheme, that produces a valid ciphertext, "must know" the corresponding plaintext.

The definition of PA was introduced by Bellare and Rogaway in [6] in the Random Oracle (RO) Model. The notion was refined in [2] and recently formulated by Bellare and Palacio [4] in the standard model. In [4] there are three main definitions of PA: each definition formalizes a stronger level of PA. They are named, in increasing order of strength, PA-0, PA-1 and PA-2.

It is important to note that PA by itself does not guarantee the secrecy of the message, so it has to be coupled with standard notions like Indistinguishability against a Chosen-Plaintext Attack (IND-CPA) or against a non-adaptive/adaptive Chosen-Ciphertext Attack (IND-CCA1, IND-CCA2). In [4] some important relations are proven, for example: PA-1+IND-CPA implies IND-CCA1 and PA-2+IND-CPA implies IND-CCA2.

PA-1 DEFINITION. The definitional framework used in [4] considers a polynomial-time adversary $\mathbf{C}$, called a ciphertext creator, that takes as input the public key and can query ciphertexts for decryption to a decryption oracle. For each such a ciphertext creator we require the existence of an *extractor* $\mathbf{C}^*$ which is another polynomial-time algorithm. $\mathbf{C}^*$ is said to be a successful extractor for $\mathbf{C}$ if it can provide replies to the oracle queries of $\mathbf{C}$ that are computational indistinguishable from those provided by a real decryption oracle. The extractor gets as input *the same public key as the ciphertext creator, as well as its coin tosses.* Basically the extractor is the "subconscious" of the adversary.

The intuition behind an extractor for PA is the following: if the only way to produce a correct ciphertext is to encrypt a plaintext then the extractor $\mathbf{C}^*$, that knows everything $\mathbf{C}$ knows, should be able to retrieve such plaintext. We now recall the formal definition from [4].

Let Dist be a distinguisher, i.e. a probabilistic polynomial time Turing machine that on input a string x outputs a bit. We define two experiments: in the first $\mathbf{C}$ interacts with the real decryption oracle dec, while on the second $\mathbf{C}$ interacts with the extractor $\mathbf{C}^*$.

Experiment $EXP^{dec}_{\mathcal{E},\mathbf{C},\mathsf{Dist}}(k)$.

$$(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{gen}(1^k)\ ;\ x \leftarrow \mathbf{C}^{\mathsf{dec}(\mathsf{sk},\cdot)}\ ;\ b \leftarrow \mathsf{Dist}(x)$$

Return b.

Experiment $EXP^{ext}_{\mathcal{E},\mathbf{C},\mathbf{C}^*,\mathsf{Dist}}(k)$.

- $(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{gen}(1^k)$;
- Choose random tapes $R(\mathbf{C})$ and $R(\mathbf{C}^*)$ for $\mathbf{C}$ and $\mathbf{C}^*$ respectively. Set $St(\mathbf{C}^*) \leftarrow [\mathsf{pk}, R(\mathbf{C})]$.
- Run $\mathbf{C}$ on input pk and random tape $R(\mathbf{C})$ until it halts. When $\mathbf{C}$ queries Q, set
$$[m, St(\mathbf{C}^*)] \leftarrow \mathbf{C}^*(Q, St(\mathbf{C}^*), R(\mathbf{C}^*))$$
and return m to $\mathbf{C}$.
- Let x denote the output of $\mathbf{C}$; Return $b \leftarrow \mathsf{Dist}(x)$.

We say that $\mathbf{C}^*$ is a successful PA-1 extractor for $\mathbf{C}$ if for any distinguisher Dist we have that

$$Prob[EXP^{dec}_{\mathcal{E},\mathbf{C},\mathsf{Dist}}(k) = 1] - Prob[EXP^{ext}_{\mathcal{E},\mathbf{C},\mathbf{C}^*,\mathsf{Dist}}(k) = 1] \le \mathsf{negl}(k)$$

We say that an encryption scheme $\mathcal{E}$ is PA-1 secure if for any ciphertext creator $\mathbf{C}$ there exists a successful PA-1 extractor $\mathbf{C}^*$.

PA-2 Definition. The PA-1 definition does not take into consideration the realistic setting in which an adversary learns valid ciphertexts by means other than encrypting a message himself: for example by eavesdropping on the network and picking ciphertexts produced by others. Given such a valid ciphertext, it may be possible for the adversary to compute another valid ciphertext, related to the original one, yet without knowing the corresponding plaintext. The PA-2 definition aims at ruling such possibility out.

When modeling the ability of receiving valid ciphertexts we need to consider two facts: (i) on the one hand $\mathbf{C}$ does not know the plaintext contained in such valid ciphertexts; (ii) on the other hand $\mathbf{C}$ may have partial information on the plaintexts and the distributions from which they are drawn. This is modeled by creating a companion *plaintext creator* $\mathbf{P}$ which when queried by $\mathbf{C}$, it generates a message m and sends it to an encryption oracle, returning the corresponding ciphertext to $\mathbf{C}$ and, in the extraction experiment, to $\mathbf{C}^*$. $\mathbf{C}$ has some control over $\mathbf{P}$ via its communication (the content of the query), but this control is not total, as the randomness of $\mathbf{P}$ is not exposed to either $\mathbf{C}$ or $\mathbf{C}^*$. Of course we do not allow $\mathbf{C}$ to query the ciphertexts created by $\mathbf{P}$ to its decryption oracle.

As before, we define two experiments: in the first $\mathbf{C}$ interacts with the real decryption oracle dec, while on the second $\mathbf{C}$ interacts with the extractor $\mathbf{C}^*$.

Experiment $EXP2^{dec}_{\mathcal{E},\mathbf{C},\mathbf{P},\mathsf{Dist}}(k)$.

- $(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{gen}(1^k)$; $L \leftarrow \epsilon$;
- Choose random tapes $R(\mathbf{C}), R(\mathbf{P})$ for $\mathbf{C}, \mathbf{P}$ respectively. Set $St(\mathbf{P}) \leftarrow \epsilon$.
- Run $\mathbf{C}$ on input pk and random tape $R(\mathbf{C})$ until it halts.
  - When $\mathbf{C}$ queries (dec, Q), if $Q \in L$ return $\perp$, o.w. return $m \leftarrow \mathsf{dec}(\mathsf{sk}, Q)$
  - When $\mathbf{C}$ queries (enc, Q), set $[m, St(\mathbf{P})] \leftarrow \mathbf{P}(Q, St(\mathbf{P}), R(\mathbf{P}))$; $c \leftarrow \mathsf{enc}(\mathsf{pk}, m)$ and $L \leftarrow L \cup \{c\}$. Return c to $\mathbf{C}$.
- Let x denote the output of $\mathbf{C}$; Return $b \leftarrow \mathsf{Dist}(x)$.

Experiment $EXP2^{ext}_{\mathcal{E},\mathbf{C},\mathbf{C}^*,\mathbf{P},\mathsf{Dist}}(k)$.

- $(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{gen}(1^k)$; $L \leftarrow \epsilon$;
- Choose random tapes $R(\mathbf{C}), R(\mathbf{C}^*), R(\mathbf{P})$ for $\mathbf{C}, \mathbf{C}^*, \mathbf{P}$ respectively. Set $St(\mathbf{P}) \leftarrow \epsilon$ and $St(\mathbf{C}^*) \leftarrow [\mathsf{pk}, R(\mathbf{C})]$.
- Run $\mathbf{C}$ on input $\mathsf{pk}$and random tape $R(\mathbf{C})$ until it halts.
  - When $\mathbf{C}$ queries (dec, Q), if $Q \in L$ return $\perp$, o.w. set $[m, St(\mathbf{C}^*)] \leftarrow \mathbf{C}^*(Q, St(\mathbf{C}^*), R(\mathbf{C}^*))$ and return m to $\mathbf{C}$.
  - When $\mathbf{C}$ queries (enc, Q), set $[m, St(\mathbf{P})] \leftarrow \mathbf{P}(Q, St(\mathbf{P}), R(\mathbf{P}))$; $c \leftarrow \mathsf{enc}(\mathsf{pk}, m)$ and $L \leftarrow L \cup \{c\}$. Return c to $\mathbf{C}$.
- Let x denote the output of $\mathbf{C}$; Return $b \leftarrow \mathsf{Dist}(x)$.

We say that $\mathbf{C}^*$ is a successful PA-2 extractor for $\mathbf{C}$ if for any plaintext creator $\mathbf{P}$ and any distinguisher Dist we have that

$$Prob[EXP2^{dec}_{\mathcal{E},\mathbf{C},\mathbf{P},\mathsf{Dist}}(k) = 1] - Prob[EXP2^{ext}_{\mathcal{E},\mathbf{C},\mathbf{C}^*,\mathbf{P},\mathsf{Dist}}(k) = 1] \leq \neg(k)$$

We say that an encryption scheme $\mathcal{E}$ is PA-2 secure if for any ciphertext creator $\mathbf{C}$ there exists a successful PA-2 extractor $\mathbf{C}^*$.

B CCA2 is not sufficient for deniability

We start by showing a CCA2 encryption scheme $\mathcal{E}$ and a secure MAC function which when used inside $\lambda_{\mathcal{E}}$ yield a protocol which is not deniable. This construction follows the original ideas from [4] where an example of a CCA2 scheme which is not plaintext-aware is presented.

Let $\mathcal{E}' = (\mathsf{gen}', \mathsf{enc}', \mathsf{dec}')$ be a CCA2-secure encryption scheme and let $f : \{0,1\}^* \rightarrow \{0,1\}^*$ be a length preserving one-way function. Given a security parameter n, consider the following new scheme $\mathcal{E} = (\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$

- key generation algorithm $\mathsf{gen}(1^n)$:
  - invokes $\mathsf{gen}'(1^n)$ to obtain public/secret keys $(\mathsf{pk}', \mathsf{sk}')$;
  - chooses at random two strings u_1, u_2 of length n and computes $U_1 = f(u_1),\ U_2 = f(u_2)$;
  - returns the public key $\mathsf{pk} = (\mathsf{pk}', U_1, U_2)$ and the secret key $\mathsf{sk} = (\mathsf{sk}', u_1, u_2)$.
- encryption algorithm $\mathsf{enc}(\mathsf{pk}, m)$:
  - parses pk as (pk', U_1, U_2);
  - returns $(0, \mathsf{enc}'(\mathsf{pk}', m)$.
- decryption algorithm $\mathsf{dec}(\mathsf{sk}, c)$:
  - parses sk as (sk', u_1, u_2) and c as (v, c');
  - if $v = 0$ then return $\mathsf{dec}'(\mathsf{sk}', c')$;
  - if $v = 1$ then if $c' = U_1|U_2$ then return $u_1|u_2$ else return *fail*.

It is not hard to see that $\mathcal{E}$ is CCA2-secure if $\mathcal{E}'$ is.

To instantiate the $\lambda_{\mathcal{E}}$ protocol we also need a suitable MAC function. Let k be a string of length n and let $\mathsf{MAC}'_k : \{0,1\}^* \to \{0,1\}^n$ be a secure MAC function. Let's define a new function $\mathsf{MAC}_k : \{0,1\}^* \to \{0,1\}^{2n}$; this function will use keys of length $2n$. Parse the key k as the concatenation of two n-bits strings such that $k = k_1|k_2$, the new MAC function is computed as $\mathsf{MAC}_{k_1|k_2}(m) = \mathsf{MAC}'_{k_1}(m)|k_2$. That is, the first half part of the key is used to compute the actual MAC function, while the second half is "published" in the output. It is easy to see that MAC is a secure MAC as long as MAC' is.

What happens if we use $\mathcal{E}$ and MAC in the $\lambda_{\mathcal{E}}$ protocol? The protocol is still a good authenticator but the deniability is gone! If the receiver uses the ciphertext $(1, U_1|U_2)$ in the first round to authenticate a message m, the sender decrypts as $\mathsf{dec}(\mathsf{sk}, (1, U_1|U_2)) = u_1|u_2$, and thus the reply will be $t = \mathsf{MAC}_{u_1|u_2}(m) = \mathsf{MAC}'_{u_1}(m)|u_2$. Notice that now the transcript contains the pre-image of the value U_2 under the one-way function f, that is the value u_2 such that $U_2 = f(u_2)$. This is evidence that the receiver has interacted with the owner of the public key containing the value U_2 (the value u_2 cannot be simulated).

C SKEME with a cooperating judge

As we discussed in Section 2 it could be possible for a judge to provide $\mathcal{M}$ with a ciphertext and ask $\mathcal{M}$ to submit it in an execution of $\lambda_{\mathcal{E}}$. In this case the protocol would stop being deniable even if the encryption is PA-2: this is because the judge *knows the decryption of the ciphertexts in the auxiliary input.*

A possible way to get around this problem is to add a challenge response mechanism (as in [27]), which however will create a non-concurrently secure 4-round protocol. Instead we show how to add a single preliminary round to $\lambda_{\mathcal{E}}$ to make it fully deniable. The resulting 3-round protocol will be simulatable (in a non black-box fashion) with auxiliary input under the assumption that $\mathcal{E}$ is PA-2 secure. More importantly it will be concurrently secure.

The basic idea is to have the sender S initiate the protocol by sending a random nonce r to the receiver R. The latter will incorporate r inside the ciphertext, i.e. will encrypt it together with the key k to produce the ciphertext c.

The sender will decrypt c and check that the plaintext starts with the nonce r and if so will MAC the message m with k, as before. If the ciphertext c is invalid or its does not start with r, then S MACs the message m with a random key. The protocol, denoted $\lambda'_{\mathcal{E}}$, is described in Figure 4.

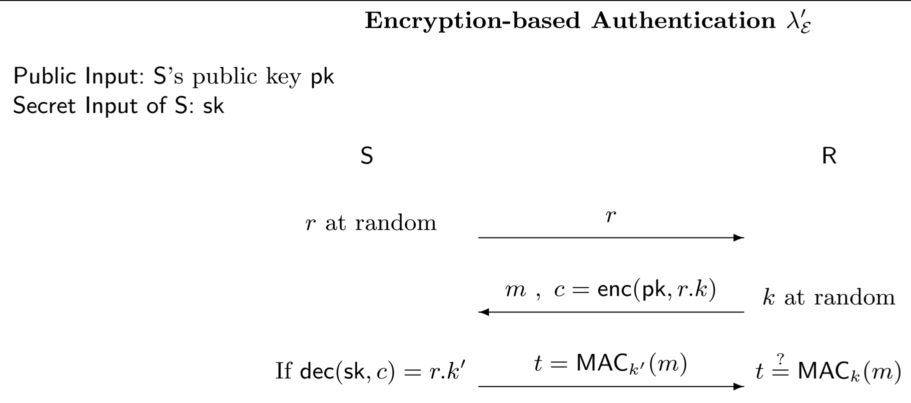
Encryption-based Authentication $\lambda'_{\mathcal{E}}$
Public Input: S's public key pk
Secret Input of S: sk
S
R
r at random
r
m , $c = \mathsf{enc}(\mathsf{pk}, r.k)$
k at random
If $\mathsf{dec}(\mathsf{sk}, c) = r.k'$
$t = \mathsf{MAC}_{k'}(m)$
$t \overset{?}{=} \mathsf{MAC}_k(m)$

Fig. 4. Augmenting $\lambda_{\mathcal{E}}$ against "cooperating judges"

To argue that $\lambda'_{\mathcal{E}}$ is deniable even in the presence of a judge who knows the decryption of the ciphertexts in the auxiliary input, we follow the same argument presented in the proof of Theorem 2. We construct a simulator $SIM_{\mathcal{M}}$ that interacting with $\mathcal{M}$, creates transcripts that are indistinguishable from real ones.

$\mathcal{M}$ can be thought of as a ciphertext creator $\mathbf{C}$ in the definition of PA-2 security. Let L be the list of ciphertexts in the auxiliary input.

We now run $\mathcal{M}$ by feeding it a random challenge r. $\mathcal{M}$ will respond with (m, c). If $c \in L$ then once again we answer with a MAC on m computed using a random key. If $c \notin L$, then by the PA-2 security assumption we have access to an extractor $\mathbf{C}^*$ which will return a value $r'.k$ as the matching plaintext contained in c. If $r \neq r'$ again we compute the MAC using a random key. Otherwise the simulator will use k to compute $t = \mathsf{MAC}_k(m)$.

This simulation is indistinguishable, because by hypothesis the answers of $\mathbf{C}^*$ when $c \notin L$ are indistinguishable from real plaintexts. On the other hand when $c \in L$ the probability that it decrypts to something of the form $r.k$ is negligible (assuming that r is sufficiently long).

SKEME KEY EXCHANGE WITH A COOPERATING JUDGE. The case of the judge who may cooperate with one of the parties in SKEME, can be handled similarly to protocol $\lambda'_{\mathcal{E}}$ (Figure 4). More specifically: we add one extra round at the beginning in which B sends g^y and nothing else. Then we use g^x and g^y as the nonce r in $\lambda'_{\mathcal{E}}$, that is we include them under the ciphertext and when decrypting each party checks that the values appear in the decrypted plaintext. As in the case of $\lambda_{\mathcal{E}'}$ this foils the attack of a judge handing a ciphertext to one of the parties in advance, since such ciphertext will include (g^x, g^y) only with negligible probability.