

Deniable Authentication on the Internet

Shaoquan Jiang

University of Calgary

Email: jiangshq@math.ucalgary.ca

Abstract. Deniable authentication is a technique that allows one party to send messages to another while the latter can not prove to a third party the fact of communication. In this paper, we first formalize a natural notion of deniable security and naturally extend the basic authenticator theorem by Bellare et al. [2] to the setting of deniable authentication. Of independent interest, this extension is achieved by defining a deniable MT-authenticator via a game. This game is essentially borrowed from the notion of universal composition [8] although we do not assume any result or background about it. Then we construct two deniable MT-authenticators: uncontrollable random oracle based and the PKI based, both of which are just 3-round protocols. The second construction assumes the receiver owns a secret key. Such a setup assumption is very popular in the real world. (Without this assumption), all the previous protocols do not have a widely satisfiable performance when applied in the Internet-like environment. Finally, as our application, we obtain key exchange protocols that is deniably secure in the real world.

1 Introduction

Authentication is a communication process, in which a receiver is assured of the authenticity of the peer identity and the authenticity of the incoming messages. This property can be guaranteed by means of a signature. Since a secure signature is unforgeable and publicly verifiable, it in other words means undeniability. This undeniability is not always desirable. For example, when you do an Internet shopping, you do not want your shopping privacy to be transferred to a third party. In this paper, we will investigate techniques for achieving deniable authentication.

1.1 Related Work

Deniable authentication was first considered in [15]. Earlier concepts occurred in [12]. Since deniability essentially requires that whatever computable through interactions is computable by adversary himself, a natural tool to achieve it is zero knowledge [20]. However, it is known that under a general complexity assumption any black-box concurrent zero-knowledge has a round complexity $\tilde{\omega}(\log \kappa)$ [33, 26, 32]. This implies that the practical deniability from it is almost impossible as most of the applications require concurrency. To overcome this barrier, [18, 16, 17, 25, 30, 13] relaxed the concurrency with a locally timing constraint. However, timing constraint is not satisfiable as it artificially increases the delay. An alternative approach is to adopt a non-standard complexity assumption. Di Rainmondo et al. [14], based on an assumption of plaintext awareness [3, 11], showed that SKEME [27] is deniably secure. But the assumption is very strong. Another alternative is to adopt a common reference string (CRS) model. In this setting, efficient concurrent zero-knowledge does exist [10, 22]. Unfortunately, it has been pointed out in the literature (e.g., Pass [31]) that deniability obtained in this way is not satisfiable as the simulator usually owns a secret of CRS while it is impossible to a real adversary. Similarly, an original random oracle [4] based solution is not satisfiable, either. Pass [31] defined a revised random oracle model (we call it an uncontrollable

random oracle (uRO) model), which is different from the original one in that the output of the oracle is maintained by an uncontrollable third party (instead of a simulator) although the simulator can still view the input-output pair of each query. Deniability under this model is practical since whatever computable by the simulator is computable by the adversary himself. However, authentication is not a research concern in [31]. As a summary, known research in deniable authentication is still not very satisfiable.

1.2 Contribution

In this paper, we first present an authentication model [2, 6] with featuring a concurrent computation model [29]. Under this, we formalize a notion of *deniable security* and naturally extend the authenticator theorem in [2] to the setting of deniable computation. This extension is essentially achieved by deploying a universal composition technique. This strategy is of independent interest. Then we construct two provably deniable MT-authenticators: uncontrollable random oracle based and PKI based, both of which are 3-round only. We remark that these results do not contradict the barriers outlined before (e.g., large round complexity). Indeed, in the previous subsection we implicitly assume that a receiver does not own a secret while in PKI-based protocol a receiver does own a secret. This setup assumption is very popular in real applications (e.g., key exchange). It is known [31] that deniability is impossible in the CRS setting. However, since participants in our protocol have secrets, our simulator has no need to use the secret for CRS. In fact, the protocol initialization in our model is executed by a third party. Thus, the simulator has no way to access the secret for CRS. Finally, as our application, we obtain a key exchange protocols that is deniably UM-secure.

2 Model

Bellare *et al.* [2, 6] formalized two models for cryptographic protocols: unauthenticated-link model (UM) and authenticated-link model (AM). This model is very useful in a modular design of UM-secure protocols. On the other hand, a concurrent composition model (e.g., [29]) is convenient in analysis of protocols. We now present UM/AM models with featuring [29].

Assume P_1, \dots, P_n are n -parties. π is an arbitrary protocol. The execution of π is modeled as follows. Each party is regarded as a polynomial time interactive Turing machine. Initially, P_i is invoked with input, identity and random input. Then he waits for an activation. P_i can be activated by incoming messages from other parties and/or external input. Once activated, P_i follows the specification of π by computing

$$\begin{aligned} &\pi(\text{input, internal state, incoming message}) \\ &= (\text{new state, outgoing messages, output}). \end{aligned}$$

Initial internal state is the party's input, identity and random input. After each activation, the internal state is updated by a new state. Each activation could generate outgoing messages (to other parties). It may also generate a local output and label the sensitive part as 'secret'. Each P_i could concurrently run many copies of π . A copy is called a *session*. Each session has a session ID. The only requirement for a session ID is its uniqueness in P_i . The input for different activations of each session in P_i might be different. It is generated by a probabilistic polynomial time algorithm Φ_i . For the ℓ th activation of the j th session, the input is $x_{\ell,j} = \Phi_i(\ell, j, x_i, \text{hist})$, where x_i is the

initial input to Φ_i and hist is the output history of all copies of π in P_i . Note $x_{\ell,j}$ could be empty. In order of delivery (also for security), *each message sent into the channel is assumed to contain (sender, sender session ID, receiver, receiver session ID)*. In addition, we implicitly assume that π has been ideally initialized by a function I : for $r \leftarrow \{0, 1\}^\kappa$, $I(r) = I(r)_0, I(r)_1, \dots, I(r)_n$, where κ is the security parameter, $I(r)_0$ is the public information known to all participants, and $I(r)_i$ is the secret key for P_i .

2.1 Unauthenticated-link Model

Roughly speaking, the unauthenticated-link model is a model for the concurrent execution of a protocol where a malicious adversary presents. In this model, the scheduling of events is completely determined by adversary \mathcal{U} . Such a scheduling consists of a sequence of activations to parties. He can activate any party P_i with an arbitrary incoming message. He can also invoke P_i to start a new session. In both cases, it is assumed that Φ_i has already supplied the input (if any). He can also delete, block, modify and insert any message over the channel. Once a party completes an activation, the outgoing message and the local output (if any), except the parts labeled as ‘secret’, are available to \mathcal{U} . \mathcal{U} can corrupt a party at any time. When one party gets corrupted, all sessions’ internal states and the secret key within this party are available to \mathcal{U} . A special note ‘corrupted’ is appended to the output of this party. It will not produce an output any more. In addition, his future action is fully taken by \mathcal{U} . \mathcal{U} can also corrupt a particular session in P_i . In this case, he obtains the current internal state for this session. A special note of corruption is appended to this session’s output. Later, it will not produce an output any more. The future execution of this session is fully taken by \mathcal{U} . We assume session corruption is possible only if the session has started. This reflects the practical concern where a session is attacked only if the attacker sees the session’s activity.

Assume the protocol is initialized by a trusted third party \mathbb{T} . Specifically, before the protocol starts, \mathbb{T} takes $s \leftarrow \{0, 1\}^\kappa$ and executes the initialization function $I(s) = \{I(s)_i\}_{i=0}^n$. Then he provides $I(s)_i$ to party P_i as his secret. The global public information is $I(s)_0$. At the end of protocol execution, \mathbb{T} outputs

$$I(s)_0 \cup \{I(s)_i \mid P_i \text{ corrupted}, 1 \leq i \leq n\}.$$

The final output of a party is defined to be the concatenation of his output history from all sessions. Let $\mathbf{x} = (x_1, \dots, x_n)$, where x_i is the initial input for Φ_i . Let $\mathbf{r} = (r_0^0, r_0^1, r_1^0, r_1^1, \dots, r_n^0, r_n^1)$ be the random input, where r_0^1 for \mathcal{U} , r_0^0 for \mathbb{T} , r_i^0 for P_i and r_i^1 for Φ_i . Let $\Phi = (\Phi_1, \dots, \Phi_n)$. We use $\text{Adv}_{\pi, \mathcal{U}, \Phi}(\mathbf{x}, \mathbf{r})$ to denote the output of \mathcal{U} , and use $\text{UnAuth}_{\pi, \mathcal{U}, \Phi}(\mathbf{x}, \mathbf{r})_i$ to denote the output of P_i . $\text{UnAuth}_{\pi, \mathcal{U}, \Phi}(\mathbf{x}, \mathbf{r})_0$ denotes the output of \mathbb{T} . Let

$$\text{UnAuth}_{\pi, \mathcal{U}, \Phi}(\mathbf{x}, \mathbf{r}) = \text{Adv}_{\pi, \mathcal{U}, \Phi}(\mathbf{x}, \mathbf{r}), \text{UnAuth}_{\pi, \mathcal{U}, \Phi}(\mathbf{x}, \mathbf{r})_0, \dots, \text{UnAuth}_{\pi, \mathcal{U}, \Phi}(\mathbf{x}, \mathbf{r})_n.$$

Let $\text{UnAuth}_{\pi, \mathcal{U}, \Phi}(\mathbf{x})$ be the random variable describing $\text{UnAuth}_{\pi, \mathcal{U}, \Phi}(\mathbf{x}, \mathbf{r})$. Our inclusion of the output of \mathbb{T} in the global output is for defining deniable security later (See Section 2.3).

2.2 Authenticated-link Model

Authenticated-link model is similar to unauthenticated-link model, except that any outgoing message sent by an uncorrupted party (if not blocked) will be faithfully delivered.

Authentication Functionality The following functionality (See Figure 1) is to formalize the authenticated channel. Unlike [8], here we directly consider the multiple sessions of the functionality.

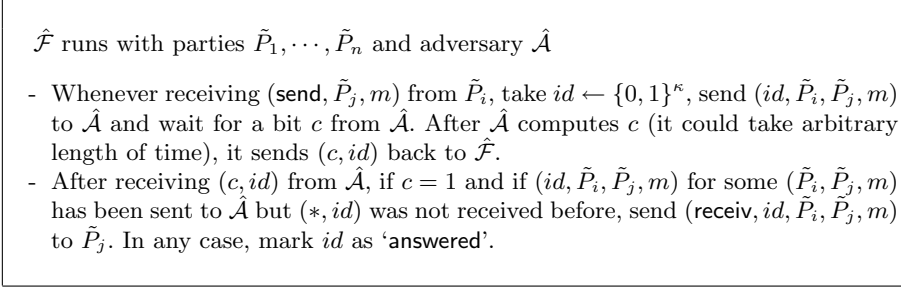


Fig. 1. Ideal functionality $\hat{\mathcal{F}}$ for authentication

This seems helpful to simplify the presentation. The action of \tilde{P}_i is defined as follows. Whenever upon input $(\text{send}, \tilde{P}_j, m)$, copy it to the input tape of $\hat{\mathcal{F}}$; whenever receiving $(\text{receiv}, id, \tilde{P}_i, \tilde{P}_j, m)$ from $\hat{\mathcal{F}}$, directly output it. The procedure (in Figure 1) for \tilde{P}_i to send m to \tilde{P}_j is called a *session* for \tilde{P}_i, \tilde{P}_j and $\hat{\mathcal{F}}$, respectively. We simply say a $\hat{\mathcal{F}}$ -session. Assume an uncorrupted sender \tilde{P}_i never sends a message m twice. This convention allows us to easily identify a replay attack. Thus, a session for an uncorrupted sender can be identified by m itself. A session in $\hat{\mathcal{F}}$ or in a receiver \tilde{P}_j can be identified by id (**for simplicity, we assume $id \leftarrow \{0, 1\}^\kappa$ never repeats in this paper**). However, when a party \tilde{P}_i is corrupted, our functionality allows $\hat{\mathcal{A}}$, in the name of \tilde{P}_i , to send any m to \tilde{P}_j (through $\hat{\mathcal{F}}$). This reflects the concern when one party is adversarial, cryptographic authentication techniques can not prevent it from flooding a receiver. Further remarks follow. First, message exchanges between \tilde{P}_i and $\hat{\mathcal{F}}$ are ideally secure and immediate (i.e., no adversary plays between). Second, $\hat{\mathcal{A}}$ can have an arbitrary delay to return (c, id) . This reflects the nature of an asynchronous network. $c = 1$ means the message m is not blocked. Third, $\hat{\mathcal{A}}$ can corrupt any \tilde{P}_i or a session in it. If a session is corrupted, the session state (i.e., input or output for this session) is provided to $\hat{\mathcal{A}}$ and a note ‘corrupted’ is appended in his output. The future action is fully taken by $\hat{\mathcal{A}}$. If \tilde{P}_i is corrupted, all the sessions in it are corrupted by $\hat{\mathcal{A}}$. In addition, the future action of \tilde{P}_i is taken by $\hat{\mathcal{A}}$. Especially, $\hat{\mathcal{A}}$ can represent \tilde{P}_i to send any message m (including a previously sent message) through $\hat{\mathcal{F}}$ to a party \tilde{P}_j .

Authenticated-Link model We are now ready to present the authenticated-link model (AM). Let P_1, \dots, P_n be n parties for executing π . AM follows the order in UM, except messages are sent/received through $\hat{\mathcal{F}}$ and the adversarial behavior is restricted correspondingly. Formally,

- When P_i needs to send a message m to P_j , it invokes \tilde{P}_i in Figure 1 with input $(\text{send}, \tilde{P}_j, m)$ to do this.
- All incoming messages for a party P_j are received through reading the output of \tilde{P}_j .
- Upon output $(\text{receiv}, id, \tilde{P}_i, \tilde{P}_j, m)$ of \tilde{P}_j , P_j executes π with incoming message m .

The action of an adversary \mathcal{A} is as follows.

- When P_i invokes \tilde{P}_i with input $(\text{send}, \tilde{P}_j, m)$, \mathcal{A} plays the role of $\hat{\mathcal{A}}$ in Figure 1 to participate.
- When a session sid in P_i is corrupted, it is assumed that all the $\hat{\mathcal{F}}$ -sessions of \tilde{P}_i that send/receive messages for session sid are corrupted. As a result, \mathcal{A} will receive the internal state of P_i in π including states from these $\hat{\mathcal{F}}$ -sessions. Finally, a noted ‘corrupted’ appears in the output of session sid . Later it is no longer active. Its action will be fully taken by \mathcal{A} .

- When a party P_i is corrupted, all sessions in P_i are corrupted. As a result, the secret key $I(r)_i$ and all internal states are available to \mathcal{A} . The future action of P_i is taken by \mathcal{A} .

The protocol is initialized by a third party \mathbb{T} . Specifically, before the protocol starts, \mathbb{T} takes $s \leftarrow \{0, 1\}^\kappa$ and executes the initialization function $I(s) = \{I(s)_i\}_{i=0}^n$. Then he provides $I(s)_i$ to party P_i . The global public information is $I(s)_0$ for all parties. In addition, \mathbb{T} can execute an extra function $I'(s') = \{I'(s')_i\}_{i=0}^n$ for $s' \leftarrow \{0, 1\}^\kappa$. Initially, $I'(s')_0$ and $I(s)_0$ will be provided to \mathcal{A} . Later whenever P_i is corrupted, $I(s)_i$ and $I'(s')_i$ will be provided to \mathcal{A} . At the end of protocol execution, \mathbb{T} outputs

$$\{I(s)_0, I(s')_0\} \cup \{I(s)_i, I'(s')_i \mid P_i \text{ corrupted}, 1 \leq i \leq n\}.$$

Note our treatment for introducing the extra $I'(s')$ is for defining deniable security (See Section 2.3), where $I'(s')$ will be the initialization function for the protocol realizing $\hat{\mathcal{F}}$. As for UM, let $\mathbf{x} = (x_1, \dots, x_n)$, where x_i is the initial input for Φ_i . Let $\mathbf{r} = (r_f, r_0^0, r_0^1, r_1^0, r_1^1, \dots, r_n^0, r_n^1)$ be the random input, where r_f is for $\hat{\mathcal{F}}$, r_0^0 is for \mathbb{T} , r_0^1 is for \mathcal{A} , r_i^0 is for P_i , r_i^1 is for Φ_i . Analogous to UM, we can define the adversary output $\text{Adv}_{\pi, \mathcal{A}, \Phi, I'}^{\hat{\mathcal{F}}}(\mathbf{x}, \mathbf{r})$, the output of \mathbb{T} $\text{Auth}_{\pi, \mathcal{A}, \Phi, I'}^{\hat{\mathcal{F}}}(\mathbf{x}, \mathbf{r})_0$, the output of party P_i $\text{Auth}_{\pi, \mathcal{A}, \Phi, I'}^{\hat{\mathcal{F}}}(\mathbf{x}, \mathbf{r})_i$, the global output $\text{Auth}_{\pi, \mathcal{A}, \Phi, I'}^{\hat{\mathcal{F}}}(\mathbf{x}, \mathbf{r})$ and the corresponding random variable $\text{Auth}_{\pi, \mathcal{A}, \Phi, I'}^{\hat{\mathcal{F}}}(\mathbf{x})$. Note in the UM case, I' is empty. Also since I is already implicitly included in π , there is no need to explicitly label it on the above variables.

2.3 Deniable Security

For a protocol π to be deniably secure, we essentially hope whatever computable by an attacker through interactions can be computed by himself alone. There are two factors to prevent a simulator from executing such a deniable computation. First, \mathbf{x} could be unknown. However, if \mathbf{x} is private, it is hard to see what type of deniability can be formalized. To simplify the problem, we only consider functionalities, where \mathbf{x} is not a secret. For instance, in key exchange, x_i is a request to carry out key exchange. One may wonder why not just define the security model such that the adversary additionally plays the role of Φ to supply protocol inputs. We stress that for some functionalities such as oblivious transfer and e-voting, the inputs are secret. Thus, the adversary is not allowed to know them unless the underlying party gets corrupted. The perfect version of security in a multi-party computation is formalized as an ideal process, where the parties hands their inputs to a trusted party who feeds back an output for each party by computing the functionality himself. Details can be found in the literature (e.g., [29]). In our setting, input \mathbf{x} is not a secret. It follows that this formulation can also be regarded as an ideal version of deniable security. Again following the multi-party tradition, the deniable security of a protocol π can be defined as requiring an ideal process simulator to simulate a real system such that the global output of the ideal process is indistinguishable to that in the real execution. However, the second problem comes. Recall that in π , each party P_i receives a secret key $I(r)_i$ from the setup function I and an adversary is unable to access an uncorrupted $I(r)_i$. Thus, in order to be deniable, a simulator should not be allowed to know uncorrupted $I(r)_i$ either. To do this, we let a third party \mathbb{T} to take $I(r) = \{I(r)_i\}$ for $r \leftarrow \{0, 1\}^\kappa$ and provide $I(r)_0$ to ideal process simulator. Later, $I(r)_i$ is provided to the latter if and only if P_i is corrupted. At the end of the simulation, \mathbb{T} output $I(r)_0$ and all corrupted $I(r)_i$. The global output in the ideal process is expanded with the output of \mathbb{T} . If π is a $\hat{\mathcal{F}}$ -hybrid protocol, then I used by \mathbb{T} in the above is replaced I and I' for an arbitrary given extra I' .

Definition 1. Let π be a protocol with initialization function I for computing a functionality \mathcal{G} . Let I' be arbitrary extra initialization function (I' is empty if π is a UM protocol). π is said to be deniably secure if for any feasible \mathbf{x}, I' and any PPT adversary \mathcal{O} against π there exists a PPT adversary \mathcal{S} against the ideal process such that

$$\text{IDEAL}_{\mathcal{G}, \mathcal{S}, \Phi, (I, I')}(\mathbf{x}) \stackrel{c}{\equiv} \text{REAL}_{\pi, \mathcal{O}, \Phi, I'}(\mathbf{x}), \quad (1)$$

where the left side and right side are the global outputs of ideal process and real process (i.e., in AM or UM) respectively.

3 Deniable Authentication Theorem

Essentially, we wish to construct a protocol ρ to realize $\hat{\mathcal{F}}$. Then for any protocol π in the $\hat{\mathcal{F}}$ -hybrid model (i.e., AM), we replace $\hat{\mathcal{F}}$ by ρ and hope the composed protocol (denoted by π^ρ) is secure. Bellare *et al.* [2] proposed a notion of MT-authenticator, which is a realization of $\hat{\mathcal{F}}$ in the UM. They confirmed the above result when ρ is a MT-authenticator. However, here we are mainly interested in finding a ρ that does not introduce undeniability. Original MT-authenticator does not guarantee this since their simulator initializes the MT-authenticator himself. In order to be deniable, a simulator should not be allowed to know the secret key of party in ρ unless he is corrupted. To achieve this, we introduce a third party \mathbb{T} to generate and maintain parties' private keys. Especially, a simulator is allowed to access $I(r)_i$ if and only if party i is corrupted. This is what we have done in the authenticated-link model. We formalize this intuition into the following notion of deniable authentication.

Definition 2. Assume ρ is a protocol for computing $\hat{\mathcal{F}}$. Let π be any protocol in the $\hat{\mathcal{F}}$ -hybrid model. Let I_ρ be the initialization function for ρ . π^ρ is said to be deniably authenticated if for any adversary \mathcal{U} against π^ρ and any \mathbf{x} , there exists an adversary \mathcal{A} against π such that

$$\text{Auth}_{\pi, \mathcal{A}, \Phi, I_\rho}^{\hat{\mathcal{F}}}(\mathbf{x}) \stackrel{c}{\equiv} \text{UnAuth}_{\pi^\rho, \mathcal{U}, \Phi}(\mathbf{x}). \quad (2)$$

Since MT-authenticator in [2] provides an authenticated transformation for any AM protocol, a question is whether there exists a natural property for ρ such that as long as ρ satisfies it π^ρ will be deniably authenticated for any π . In the following, we introduce a notion of deniable MT-authenticator. We show that given a deniable MT-authenticator ρ , for any π in the $\hat{\mathcal{F}}$ -hybrid model, π^ρ is deniably authenticated. We define this notion through two protocol games. These game are essentially borrowed from the notion of universal composition [8] although we do not need any result or background about it.

The first game is denoted by G_0 . Assume $\tilde{P}_1, \dots, \tilde{P}_n$ is running ρ in the UM with a dummy adversary \mathcal{A}_0 . \mathcal{Z} is a PPT interactive Turing machine. Assume $I(r) = \{I(r)_i\}_{i=0}^n$ is the initialization function for ρ . Initially, \mathcal{Z} will receive the public information $I(r)_0$. On the one hand, \mathcal{Z} plays the role of Φ to supply inputs for \tilde{P}_i . On the other hand, \mathcal{Z} can supply \mathcal{A}_0 with instructions obeying the UM rules at any time. These instructions include (1) starting a session at some P_i or (2) activating a session with a specific incoming message or (3) corrupting a session or a party. Upon an instruction, \mathcal{A}_0 executes it faithfully. In case of (1)(2), \mathcal{A}_0 outputs the outgoing message (if any) generated by the activated session. In case of (3), \mathcal{A}_0 outputs the collected information. \mathcal{Z} can read the outputs from all parties including \mathcal{A}_0 . At the end of the game (which is decided by \mathcal{Z}), \mathcal{Z} outputs a bit b' . This completes the description of G_0 . Now we define the second game. Denote

it by G_1 . Assume $\tilde{P}_1, \dots, \tilde{P}_n$ is executing $\hat{\mathcal{F}}$ with an adversary \mathcal{A}_1 . A PPT machine \mathcal{Z} is described as follows. Initially, a third party \mathbb{T}_ρ takes $I(r) = \{I(r)_i\}_{i=0}^n$ for $r \leftarrow \{0, 1\}^\kappa$ and provides $I(r)_0$ to both \mathcal{A}_1 and \mathcal{Z} . Later $I(r)_i$ is provided to \mathcal{A}_0 if \tilde{P}_i is corrupted. The remaining description for \mathcal{Z} (i.e., supplying inputs to \tilde{P}_i and instructions to \mathcal{A}_1) is exactly as in G_0 , except \mathcal{A}_1 instead of \mathcal{A}_0 will respond to these instructions. The action of \mathcal{A}_1 is arbitrary, except that he follows the rules of ideal process in computing $\hat{\mathcal{F}}$ (see Section 2.2). At the end of G_1 , \mathcal{Z} generates a bit b' . Now we are ready to define our notion of deniably secure MT-authenticator.

Definition 3. *Let ρ be a protocol for computing $\hat{\mathcal{F}}$. ρ is a deniable MT-authenticator if there exists a PPT simulator \mathcal{A}_1 such that for every PPT machine \mathcal{Z} ,*

$$\Pr[\mathcal{Z}(G_0) = 1] - \Pr[\mathcal{Z}(G_1) = 1] \quad (3)$$

is negligible.

Essentially, G_b is placed in a black box. The task of \mathcal{Z} is to guess which game is inside. We show if no \mathcal{Z} can win significantly better than a trivial guess, then such a ρ gives rise to a deniable transformation for any $\hat{\mathcal{F}}$ -hybrid protocol.

Theorem 1. *If ρ is a deniable MT-authenticator and π is a protocol in the $\hat{\mathcal{F}}$ -hybrid model, then π^ρ is deniably authenticated.*

Before our actual proof, we first present the main idea. Essentially, \mathcal{A} follows \mathcal{U} , except the part in executing $\hat{\mathcal{F}}$, where \mathcal{A} activates \mathcal{A}_1 through a sequence of instructions. If the ideal process in executing $\hat{\mathcal{F}}$ with \mathcal{A}_1 is replaced with the real execution of ρ with \mathcal{A}_0 , then \mathcal{A} becomes identical to \mathcal{U} . Thus, if (2) is violated, we can construct a PPT machine \mathcal{Z}_ρ to distinguish G_0 and G_1 as follows. \mathcal{Z}_ρ simulates $\{P_i\}$ in π and Φ , and also follows \mathcal{A} , except that whenever he needs to simulate a message transmission, he does it through the challenge G_b for $b \leftarrow \{0, 1\}$. Finally, \mathcal{Z}_ρ provides the simulated global output to a distinguisher and outputs whenever he does. If $b = 1$, the global output in the simulation of \mathcal{Z}_ρ is distributed as $\text{Auth}_{\pi, \mathcal{A}, \Phi, I_\rho}^{\hat{\mathcal{F}}}(\mathbf{x})$; otherwise, it is distributed as $\text{UnAuth}_{\pi, \mathcal{U}, \Phi}(\mathbf{x})$. As a result, violation of (2) implies a non-negligible advantage of \mathcal{Z}_ρ , contradicting the assumption of ρ . Now we start to implement the above idea.

Proof. Let \mathcal{U} be against π^ρ . Assume $I(r) = \{I(r)_i\}_{i=0}^n$ and $I_\rho(r') = \{I_\rho(r')_i\}_{i=0}^n$ be the initialization function for π and ρ respectively. With the above idea in mind, we first construct \mathcal{A} against π in the $\hat{\mathcal{F}}$ -hybrid model.

- a. \mathcal{A} receives $I(r)_0, I_\rho(r')_0$ respectively for π and ρ from \mathbb{T} , invokes \mathcal{U} with $I(r)_0$ and $I_\rho(r')_0$. P_i will receive $I(r)_i$ and $I(r)_0$ from \mathbb{T} . In addition, \mathcal{A} initializes \mathcal{A}_1 with $I_\rho(r')_0$.
- b. Whenever P_i wishes to send m to P_j , he plays the role of \tilde{P}_i (in Figure 1) to send $(\text{send}, \tilde{P}_j, m)$ to $\hat{\mathcal{F}}$, who will take $id \leftarrow \{0, 1\}^\kappa$ and send $(id, \tilde{P}_i, \tilde{P}_j, m)$ to \mathcal{A} . \mathcal{A} then, in the role of $\hat{\mathcal{F}}$, activates \mathcal{A}_1 with $(id, \tilde{P}_i, \tilde{P}_j, m)$. After seeing any output by \mathcal{A}_1 , forward it to \mathcal{U} .
- c. Whenever \mathcal{U} requests to deliver a message msg to P_j , activate \mathcal{A}_1 with this request. In turn, if \mathcal{A}_1 generates any output msg' , \mathcal{A} provides it to \mathcal{U} . (Remark: as the output of \mathcal{A}_0 in such a query is to report the outgoing message from P_j , the output \mathcal{A}_1 is supposed to be a simulation of such a message.) If \mathcal{A}_1 generates an outgoing message (c, id) to $\hat{\mathcal{F}}$, \mathcal{A} sends (c, id) to $\hat{\mathcal{F}}$ as his reply for the request of bit c .

- d. Whenever \mathcal{U} requests to see an output of party i , collect the output of P_i in π (not including the parts labeled as ‘secret’) and provide it to \mathcal{U} . Note since both \mathcal{A} and \mathcal{U} are not allowed to see the secret parts, this simulation is perfect.
- e. Whenever \mathcal{U} asks to corrupt a session id in P_i , corrupt the corresponding session in π and obtain the session state $stat$. In addition, he, the role of \mathcal{Z} in G_1 , requests \mathcal{A}_1 to corrupt all the sessions that sending messages from session id (Recall each message contains the sender session ID of π ; See the protocol syntax in Section 2). As results, \mathcal{A}_1 will output simulated internal states $stat'$ for all these sessions. \mathcal{A} merges $stat'$ and $stat$ to provide a complete session state st for session id of P_i in π . Finally, \mathcal{A} provides st to \mathcal{U} .
- f. Whenever \mathcal{U} asks to corrupt a party P_i , corrupt P_i in π to get $I(r)_i$ and then obtain the secret key $I_\rho(r)_i$ (from \mathbb{T} by requesting \mathcal{A}_1 to corrupt \tilde{P}_i). Obtain internal states for all sessions in P_i through session corruption as in item (e). Finally, provide all the information to \mathcal{U} .

Finally, \mathcal{A} outputs whatever \mathcal{U} does.

We claim that \mathcal{A} will satisfy (2). Otherwise, we construct a PPT machine \mathcal{Z}_ρ to distinguish G_0/G_1 with a non-negligible advantage. To do this, we first show that the simulation of \mathcal{A} can be completed by black-box access to the game G_1 . Indeed, we only need to check the black-box access restriction for \mathcal{A} can be satisfied.

- In item (a), this will be violated when \mathcal{A} initializes \mathcal{A}_1 with $I_\rho(r')_0$. However, since \mathbb{T} already initializes \mathcal{A}_1 with it, this operation is not required any more.
- In item (b), the code exactly follows the description of G_1 . Thus, \mathcal{A} only needs to feed input $(\text{send}, \tilde{P}_j, m)$ and read the output from \mathcal{A}_1 .
- In item (c), \mathcal{A} only needs to feed the instruction “deliver message msg to P_j ” to \mathcal{A}_1 . The remaining computation will be perfectly simulated in G_1 .
- Item (d)(e)(f) do not violate black-box restriction.

This revision does not change the global output of the simulation (i.e., $\text{Auth}_{\pi, \mathcal{A}, \Phi, I_\rho}^{\hat{\mathcal{F}}}(\mathbf{x})$). On the other hand, when the black-box G_1 is replaced with G_0 , then the global output of the simulated game is distributed exactly as $\text{UnAuth}_{\pi, \mathcal{U}, \Phi}(\mathbf{x})$. Now we are ready to describe the code of \mathcal{Z}_ρ . Given black-box G_b , auxiliary input \mathbf{x} and $I_\rho(r')_0$, he initializes $\{I(r)_i\}$ for π in $\hat{\mathcal{F}}$ -hybrid mode, simulates $\{P_i\}$ faithfully, plays the role of Φ , and also follow the revised \mathcal{A} with black-box access to G_b . Finally, \mathcal{Z}_ρ provides the global output out of the simulated game to the distinguisher of equation (2) and outputs whatever he does. Note that if $b = 0$, then out is distributed according to the right hand of equation (2); the right hand of (2) otherwise. Thus, non-negligible advantage of the distinguisher implies non-negligible advantage of \mathcal{Z}_ρ , contradiction to assumption on ρ . ■

Corollary 1. *Assume ρ is deniably MT-authenticator and π is deniably secure in the $\hat{\mathcal{F}}$ -hybrid model, then ρ is deniably secure in the UM.*

4 Our Deniable MT-Authenticators

4.1 Uncontrollable Random Oracle Based Deniable MT-Authenticator

In this subsection, we will construct a deniable MT-authenticator from random oracle. Notice that the original random oracle [4] is completely controlled by a simulator. Especially, if an oracle query is issued by the simulator himself, he can first choose the output and then decide the query input.

This provides too much freedom to a simulator. As pointed out by Pass [31], a solution obtained in this way is *not* deniable as a real attacker does not have this right at all. The random oracle we use here is defined by Pass. This object is maintained by an uncorruptible third party but all the input-output pairs are seen by the simulator. We call it *Uncontrollable Random Oracle* (uRO). Deniability makes sense in this model since whatever computable by a simulator is computable by an attacker himself.

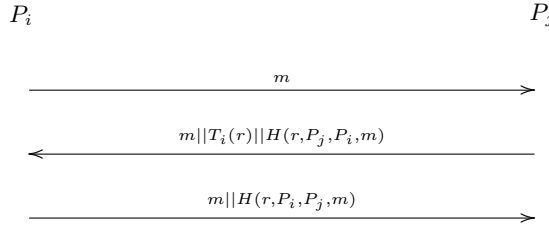


Fig. 2. Our Deniable MT-Authenticator uRO-Auth
(Note the complete details appear in the context)

Now we describe our uRO based MT-authenticator. We call it uRO-Auth MT-authenticator. Assume P_i wishes to send a message m to P_j . Let T_i be the public-key of a trapdoor permutation owned by party P_i and D_i be the trapdoor for T_i . P_i first sends m to P_j . P_j then takes $r \leftarrow \{0, 1\}^\kappa$, computes and sends back $m || T_i(r) || H(r, P_j, P_i, m)$ to P_i . Receiving $m || \alpha || \beta$, P_i computes $r' = D_i(\alpha)$. If $r' \neq \perp$, it checks whether $\beta = H(r', P_j, P_i, m)$. If yes, he computes and sends out $m || \gamma$ to P_j , where $\gamma = H(r', P_i, P_j, m)$. If $r' = \perp$ or β is not successfully verified, he does nothing. Upon receiving $m || \gamma$, P_j checks whether $\gamma = H(r, P_i, P_j, m)$. If yes, he generates a local output “(receiv, id , P_i , P_j , m)” for $id \leftarrow \{0, 1\}^\kappa$; otherwise, it does nothing. The graphic interpretation of the protocol is presented in Figure 2.

In the following, we show that uRO-Auth is a deniable MT-authenticator. We first outline the proof strategy, which is quite general and will be used throughout this paper. Assume we generally need to prove a protocol ρ is a deniable MT-authenticator. Then we need to construct a simulator \mathcal{A}_1 such that Definition 3 is satisfied. On the one hand, we let \mathcal{A}_1 participate in the ideal process for $\hat{\mathcal{F}}$. On the other hand, \mathcal{A}_1 simulates P_1, \dots, P_n to execute ρ , where the adversarial behaviors are described by the instructions from \mathcal{Z} . \mathcal{A}_1 uses the simulated ρ to help compute c (requested by $\hat{\mathcal{F}}$ in the ideal process). The simulation is so good such that the global outputs in the real ρ and in the ideal process of $\hat{\mathcal{F}}$ are indistinguishable. As \mathcal{Z} only read the output of G_b ($b = 0, 1$), he only has a negligible advantage in distinguishing G_0/G_1 .

Theorem 2. *If H is an uRO, then uRO-Auth is a deniable MT-authenticator.*

Proof. Keep the notations as in the definition of deniable MT-authenticator. We need to construct a simulator \mathcal{A}_1 such that for any PPT machine \mathcal{Z}

$$\Pr[\mathcal{Z}(G_0) = 1] - \Pr[\mathcal{Z}(G_1) = 1] \tag{4}$$

is negligible. The code of \mathcal{A}_1 is as follows. First of all, \mathbb{T} randomly samples $\{(T_i, D_i)\}$ and provides $\{T_i\}$ to both \mathcal{Z} and \mathcal{A}_1 . The uncontrollable random oracle H is assumed to work as follows. It

maintains a list L_H which is initially empty. Upon a hash query x , this H -oracle checks whether x was queried before. If not, it takes $y \leftarrow \{0, 1\}^\kappa$ and adds (x, y) to L_H ; otherwise, it takes the existing record (x, y) from L_H . The answer to query x is y . The detailed simulation by \mathcal{A}_1 is as follows.

- I₁ Whenever \mathcal{A}_1 receives a message $(id, \tilde{P}_i, \tilde{P}_j, m)$ (from $\hat{\mathcal{F}}$) and is asked for a bit c , he internally simulates P_i to send a flow one message m to P_j in his simulated uRO-Auth and reports this flow one message to \mathcal{Z} (intuitively, let \mathcal{Z} believe he is interacting with real execution of uRO-Auth).
- I'₁ Whenever \mathcal{A}_1 was requested to start a session in the name of corrupted P_i to authenticate a message m to P_j , \mathcal{A}_1 first in the name of corrupted \tilde{P}_i sends $(\text{send}, \tilde{P}_j, m)$ to $\hat{\mathcal{F}}$. The remaining action of this query is to follow item I₁.
- I₂ Whenever \mathcal{Z} requests \mathcal{A}_1 to deliver a message msg from P_i to a responder P_j (i.e., P_j plays the authentication receiver in the protocol), \mathcal{A}_1 does the following. \mathcal{A}_1 represents P_j to do so honestly in the simulation of uRO-Auth. If msg is a flow one message, he reports the simulated flow two message back to \mathcal{Z} ; otherwise, msg is flow three message. In this case, if the simulated P_j accepts, $c = 1$; $c = 0$ otherwise. Feedback (c, id) to $\hat{\mathcal{F}}$, where if some $(id, \tilde{P}_i, \tilde{P}_j, m)$ was received from $\hat{\mathcal{F}}$ but $(*, id)$ has not been feedback, take id as in this received tuple; otherwise $id \leftarrow \{0, 1\}^\kappa$. In any case, if $c = 1$, \mathcal{A}_1 simulates P_j to generate an output $(\text{receiv}, id, P_i, P_j, m)$. Denote the event that $c = 1$ but $(id, \tilde{P}_i, \tilde{P}_j, m)$ was never received before by Bad_0 ; denote the event \tilde{P}_i is uncorrupted and $c = 1$ but $(c^*, *)$ on m for a bit c^* was sent to $\hat{\mathcal{F}}$ by Bad_1 . Note under $\neg\text{Bad}_0 \wedge \neg\text{Bad}_1$, outputs of P_j and \tilde{P}_j are identical.
- I₃ Whenever \mathcal{A}_1 is asked to deliver a Flow2 message $m||\alpha||\beta$ to P_i , \mathcal{A}_1 checks whether L_H has a record $((r', P_j, P_i, m), \beta)$ for some r' such that $\alpha = T_i(r')$. If the check fails, it terminates this session; otherwise (in this case r' is unique since T_i is a permutation), asks H -oracle for query (r', P_i, P_j, m) . Assume the answer is γ . He then simulates to send out $m||\gamma$ to P_j and reports this message to \mathcal{Z} . This simulation is perfect except when β happens to be valid while (r', P_i, P_j, m) satisfying $\alpha = T_i(r')$ was not queried to H -oracle (this covers the attack by forging flow two message without query (r', P_i, P_j, m) to H -oracle). We note this event by \mathbf{E}_1 . We know that the number of Flow2 message is upper bounded by run time of \mathcal{Z} (denoted by R_z). Then, $\Pr[\mathbf{E}_1] \leq \frac{R_z}{2^\kappa}$.
- I₄ Whenever \mathcal{A}_1 is requested to reveal a session in P_t , \mathcal{A}_1 represents P_t to do so honestly. No matter P_t is a sender or receiver of m , we have that before Flow2 message the session state of P_t is m while after Flow2 message the session state of P_t is $m||r'$. Note the above session state is well defined if event $\neg\mathbf{E}_1$ holds. \mathcal{A}_1 then reports the collected session state back to \mathcal{Z} .
- I₅ When \mathcal{A}_1 is requested to corrupt P_t , he first obtains D_t from \mathbb{T} , then combines all the internal states in sessions in P_t . Finally report them to \mathcal{Z} . This simulation is perfect under event $\neg\mathbf{E}_1$.

From the above simulation, under $\neg\text{Bad}_0 \wedge \neg\text{Bad}_1$, the outputs of P_i and \tilde{P}_i are exactly identical. In addition, the simulation of \mathcal{A}_1 differs from the real execution of uRO-Auth only if \mathbf{E}_1 occurs. Thus, under $\neg\text{Bad}_0 \wedge \neg\text{Bad}_1 \wedge \neg\mathbf{E}_1$, the view of \mathcal{Z} is identical to when interacting with G_0 . So it remains to show that $\Pr[\text{Bad}_0 \vee \text{Bad}_1 \vee \mathbf{E}_1]$ is negligible. First, Bad_0 occurs if uncorrupted P_j successfully verifies a flow three message (m, γ) and thus attempts to feedback a bit (c, id) to \mathcal{F} but he never received a $(id, \tilde{P}_i, \tilde{P}_j, m)$ from the latter. Bad_1 implies two uncorrupted sessions accepts m . Since no uncorrupted sender sends the same m twice, at least one session has no sender session. Thus, Bad_1 occurs only if r taken in these two receiver sessions happen to be identical or otherwise if (r, P_i, P_j, m, γ) with different r are consistent for both sessions (which has a probability $\frac{R_z^2}{2^\kappa}$, as for at least one session (r, P_i, P_j, m) was not queried to H -oracle prior to receipt of γ). This gives

$\Pr[\text{Bad}_1] \leq \frac{2R_z^2}{2^\kappa}$. We now bound $\Pr[\text{Bad}_0 \wedge \neg \text{E}_1]$. Let ϵ be the probability that a trapdoor permutation adversary succeeds within run time R_z . We show that $\Pr[\text{Bad}_0 \wedge \neg \text{E}_1] \leq nR_z\epsilon$. Intuitively, a $\text{Bad}_0 \wedge \neg \text{E}_1$ implies \mathcal{Z} is able to decrypt the permutation in flow two in order to forge a valid flow three message. Details follow.

Consider a trapdoor permutation adversary \mathcal{I} who takes $\ell \leftarrow \{1, \dots, n\}$. Upon receiving the challenge trapdoor public-key T and a permutation challenge W , he runs \mathcal{Z} and plays the code of $\hat{\mathcal{F}}$ and \mathcal{A}_1 interacting with it, except that T_ℓ is defined to be T . Note the number of the receiver sessions is bounded by R_z . \mathcal{I} takes $L \leftarrow \{1, \dots, R_z\}$, hoping that Bad_0 will occur to the L th receiver session. The simulation code of \mathcal{I} is to play the roles of \mathcal{A}_1 and $\hat{\mathcal{F}}$, except the followings.

- a. Whenever seeing that a query $(r, P_a, P_b, *)$ for any a, b is sent to H -oracle, \mathcal{I} first checks whether $T_\ell(r) = W$. If yes, it succeeds and terminates; otherwise, it does nothing.
- b. when \mathcal{I} is requested to activate the L th receiver session (say at party P_v) with incoming message m^* , \mathcal{I} checks whether the sender implied in m^* is the owner of T_ℓ . If yes, it takes $\beta^* \leftarrow \{0, 1\}^\kappa$ and simulates to send $m^* || W || \beta^*$ to P_ℓ and reports this message to \mathcal{Z} ; otherwise, he aborts (this implies that the choice of (ℓ, L) is wrong).
- c. when \mathcal{Z} requests to corrupt P_ℓ , or reveal the L th receiver session or its corresponding sender session, \mathcal{I} is unable to provide D_z for the first case and is unable to provide the state r for the remaining two cases. He aborts the simulation. Note since Bad_0 never happens to a corrupted sender or revealed session, this abortion event implies the guess of (ℓ, L) is wrong.
- d. when \mathcal{I} is requested to deliver a Flow2 message $m' || W || \beta'$ to P_j , item (a) guarantees that $(D_\ell(W), P_a, P_b, *)$ was not queried to H -oracle. If $m' = m^*$ and $\beta' = \beta^*$, then \mathcal{I} takes $\gamma^* \leftarrow \{0, 1\}^\kappa$ and simulates to send out $m^* || \gamma^*$ to P_v and reports this message to \mathcal{Z} . Otherwise, P_ℓ simply rejects and terminates this session. Note the wrong reject implies a E_1 event. Thus, under event $\neg \text{E}_1$, the simulation is perfect.
- e. when the L th receiver session (in party P_v) is activated with $m^* | \gamma'$ later, \mathcal{I} checks if there is any $((r, P_z, P_v, m^*), \gamma')$ in L_H such that $T_z(r) = W$. If yes, it succeeds with r ; otherwise, if $\gamma' = \gamma^*$ (if γ^* has been defined), P_v generates a local output “(receiv, id , P_z , P_v , m)” where take $id \leftarrow \{0, 1\}^\kappa$ if it does not exist, otherwise, it simply rejects. Note the wrong reject occurs, denoted by event E'_1 , with probability $\leq \frac{R_z}{2^\kappa}$.

Let us denote the global output when $\text{Bad}_0 \wedge \neg \text{E}_1 \wedge \neg \text{E}'_1$ happens to (ℓ, L) by $\Pi_{\ell, L}$, denote the global output when $\text{E}_1 \vee \text{E}'_1$ occurs by Π_1 , denote the global output when neither Bad_0 nor E_1 nor E'_1 occurs by Π_∞ . Note before an abortion event occurs, the simulation under $\neg \text{E}_1 \wedge \neg \text{E}'_1$ is perfect as by \mathcal{A}_1 . Therefore, we have

$$\begin{aligned} \Pr[\text{Succ}(\mathcal{I})] &= \frac{1}{nR_z} \sum_{\ell, L} \Pr[x \in \Pi_{\ell, L}] \\ &\geq \frac{1}{nR_z} \Pr[x \in \cup_{\ell, L} \Pi_{\ell, L}] \\ &= \frac{1}{nR_z} \Pr[\text{Bad}_0 \wedge \neg \text{E}_1 \wedge \neg \text{E}'_1], \end{aligned}$$

Thus, $\Pr[\text{Bad}_0 \wedge \neg \text{E}_1 \wedge \neg \text{E}'_1] \leq nR_z\epsilon$. This implies that $\Pr[\text{Bad}_0 \wedge \neg \text{E}_1] \leq nR_z\epsilon + \frac{R_z}{2^\kappa}$.

Since $\Pr[\text{Bad}_0 \vee \text{E}_1 \vee \text{Bad}_1] \leq \Pr[\text{Bad}_0 \wedge \neg \text{E}_1] + \Pr[\text{E}_1] + \Pr[\text{Bad}_1] \leq nR_z\epsilon + \frac{2R_z + 2R_z^2}{2^\kappa}$, we have $\Pr[\mathcal{Z}(\text{G}_1) = 1] - \Pr[\mathcal{Z}(\text{G}_0) = 1] \leq nR_z\epsilon + \frac{2R_z + 2R_z^2}{2^\kappa}$ too. This concludes our proof. \blacksquare

4.2 PKI-based Deniable MT-Authenticator

PKI-based deniability is rarely studied in the literature. However, deniability in this case is very important as lots of protocols (e.g. key exchange) are built under PKI. In this subsection, we will

construct a deniable MT-Authenticator based on PKI. An immediate idea is to directly use ring signatures [34]. However, this solution is not fully deniable as one can conclude one of ring members signed the message. In the following, we introduce our new construction.

Let $(Gen(1^\kappa), E, D)$ be a public-key encryption system, where $Gen(1^\kappa)$ is the key generation algorithm for encryption/decryption key pair (ek, dk) and E/D are encryption/decryption algorithms. Let $(G(1^\kappa), RS, RV)$ be a ring signature scheme, where $G(1^\kappa)$ is the signing/public key pair (sk, vk) and S/V are signing/verification algorithms. For a user P_i , the PKI authority takes $(ek_i, dk_i) \leftarrow Gen(1^\kappa)$, $(sk_i, vk_i) \leftarrow G(1^\kappa)$. The certificate for P_i is the authority's digital signature on (ed_i, vk_i) . Let the global public-key $I_0 = \{ek_i, vk_i\}_{i=1}^n$. Each user will receive his private key (sk_i, dk_i) . For simplicity, we use E_j to represent the encryption using ek_j . When the ring of two users P_i and P_j is clear, we use $RS_i()$ to represent the signing process with key sk_i and simply use $RV_{ij}()$ to represent the verification algorithm for the ring P_i and P_j . In addition, assume \mathcal{P}_{ij} is a non-interactive zero knowledge protocol for relation $\mathcal{R} = \{\langle k \| u \| u', E_i(k; u) \| E_j(k; u') \rangle : k, u, u' \in \{0, 1\}^\kappa\}$, where $E(x; y)$ is an encryption of x using randomness y . We also use \mathcal{P} to denote such a protocol where P_i and P_j are unspecified. We use $\mathcal{P}_{ij} \{E_i(k; u) \| E_j(k; u'); k \| u \| u'\}$ to denote a random proof for an input $(E_i(k; u), E_j(k; u'))$. Our PKI-Auth MT-authenticator is as follows.

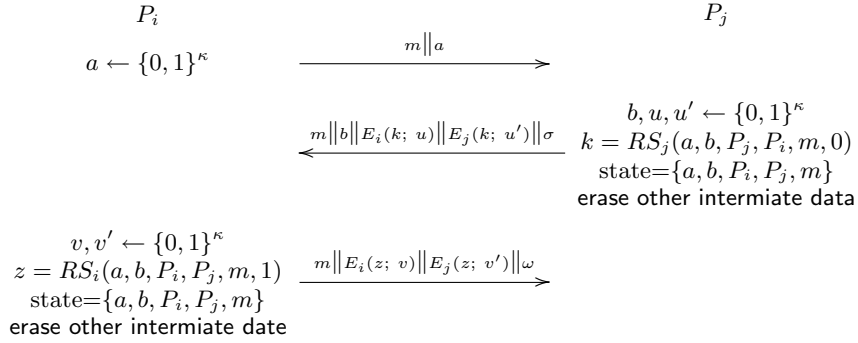


Fig. 3. Our Deniable MT-Authenticator PKI-Auth
(Note the complete details appear in the context)

1. When given input (send, P_j, m), P_i takes $a \leftarrow \{0, 1\}^\kappa$, sends out $m \| a$.
2. P_j takes $b, u, u' \leftarrow \{0, 1\}^\kappa$, computes and sends

$$m \| b \| E_i(k; u) \| E_j(k; u') \| \sigma \quad (5)$$

to P_i , where $k = RS_j(a, b, P_j, P_i, m, 0)$ and $\sigma \leftarrow \mathcal{P}_{ij} \{E_i(k; u) \| E_j(k; u'); k \| u \| u'\}$. In addition, P_j defines the internal state to be (a, b, P_i, P_j, m) and **erases** other intermediate data.

3. P_i verifies whether σ is valid, decrypts k from $E_i(k; u)$ and checks whether $RV_{ij}((a, b, P_j, P_i, m, 0), k) = 1$. Next, he takes $v, v' \leftarrow \{0, 1\}^\kappa$, computes and sends

$$m \| E_i(z; v) \| E_j(z; v') \| \omega \quad (6)$$

to P_j , where $z = RS_i(a, b, P_i, P_j, m, 1)$ if Flow2 is successfully verified, $z \leftarrow \{0, 1\}^\kappa$ otherwise, and $\omega \leftarrow \mathcal{P}_{ij} \{E_i(z; v) \| E_j(z; v'); z \| v \| v'\}$. In addition, P_j **erases** all the intermediate data and terminates.

4. P_j verifies whether ω is valid, decrypts z from $E_j(z; v')$ and checks whether $RV_{ij}((a, b, P_i, P_j, m, 1), z) = 1$. If the verifications are successful, then P_j generates a local output “(receiv, id, P_i, P_j, m)” for $id \leftarrow \{0, 1\}^\kappa$ and terminates; otherwise, it does nothing.

In the following, we show that our PKI-Auth protocol is a deniable MT-authenticator against a non-adaptive adversary. To do this, we need \mathcal{P} to be an adaptive non-interactive zero-knowledge proof for relation \mathcal{R} , where ‘adaptive’ means the system parameter is chosen before the common input to be proven is chosen; See [28]. The proof of Theorem is put in appendix.

Theorem 3. *Assume \mathcal{P} is an adaptive non-interactive zero-knowledge proof for relation \mathcal{R} with negligible soundness error. $(Gen(1^\kappa), E, D)$ is a CCA2 secure public-key encryption scheme, and (RS, VK) is an existentially unforgeable ring signature scheme. Then PKI-Auth is a deniable MT-authenticator against non-adaptive adversary.*

5 Application to Deniable Key Exchange

Key exchange is a communication procedure in which participants establish a temporarily shared secret key. To evaluate the security, several models are proposed in the literature [2, 4, 6]. Here we use the model in [24], a slightly revised version of [2]. In this model, an ideal process is defined. Then a real protocol λ is constructed. λ is said to be secure if for any adversary against λ , there exists an adversary against the ideal process such that the global output in these two worlds are indistinguishable. Here the ideal process as well as the security definition should be slightly modified to be consistent with that in Section 2.3. In [24], a \mathcal{F} -hybrid secure key exchange protocol Encr-KE was proposed (See Figure 4), where $(\mathcal{G}(1^\kappa), \mathcal{E}, \mathcal{D})$ is a semantically secure public-key encryption scheme. Notice that this protocol has an empty initialization function. It follows that Encr-KE is deniably secure in the \mathcal{F} -hybrid model in the sense of Definition 1 (note the original proof needs to be slightly modified in order to cater our formalization of authenticated-link model).

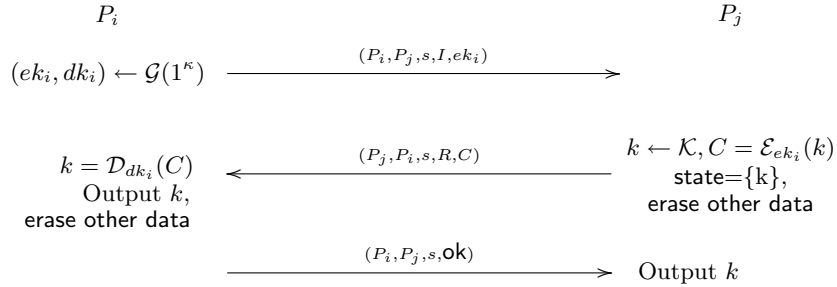


Fig. 4. AM-secure Key Exchange Protocol Encr-KE, Details see [24]

Denote the key exchange protocol obtained by applying uRO-Auth and PKI-Auth to Encr-KE by uROE-KE and PKIE-KE, respectively. From the deniable authenticator theorem, we have

Theorem 4. *uROE-KE and PKIE-KE are deniably secure key exchange protocols in the UM, where the former is adaptively secure while the latter is only non-adaptively secure.*

References

1. M. Bellare, A. Boldyreva and S. Micali, Public-Key Encryption in a Multi-User Setting: Security Proofs and Improvements, *EUROCRYPT'00*, 2000.
2. M. Bellare, R. Canetti, and H. Krawczyk, a modular approach to the design and analysis of authentication and key exchange protocols, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pp. 419-428, 1998, Dallas, Texas, USA.
3. M. Bellare and A. Palacio, Towards Plaintext-Aware Public-Key Encryption without Random Oracles, *Advances in Cryptology-ASIACRYPT'04*, Springer-Verlag, 2004.
4. M. Bellare and P. Rogaway, Entity authentication and key distribution, *Advances in Cryptology-CRYPTO'93*, D. Stinson (Ed.), LNCS 773, Springer-Verlag, 1993.
5. M. Bellare and P. Rogaway, Random Oracle is Practical: A Paradigm for Designing Efficient Protocols, *ACM CCS'93*, pp. 62-73.
6. R. Canetti and H. Krawczyk, analysis of key-exchange protocols and their use for building secure channels, *Advances in Cryptology-EUROCRYPT 2001*, B. Pfitzmann (Ed.), LNCS 2045, Springer-Verlag, pp. 453-474, 2001.
7. R. Canetti, Security and Composition of Multi-party Cryptographic Protocols, *Journal of Cryptography*, Vol. 13, No. 1, pp. 143-202, 2000.
8. R. Canetti, Universally Composable Security: A New Paradigm for Cryptographic Protocols, *FOCS'01*, 2001.
9. R. Cramer and V. Shoup, a practical public-key cryptosystem provably secure against adaptive chosen ciphertext attack, *Advances in Cryptology-CRYPTO 1998*, H. Krawczyk (Ed.), LNCS 1462, Springer-Verlag, pp. 13-25, 1998.
10. I. Dămgård, Efficient Concurrent Zero-Knowledge in the Auxiliary String Model, *Advances in Cryptology-Eurocrypt 2000*, pp. 418-430, 2005.
11. A. Dent, The Cramer-Shoup Encryption Scheme is Plaintext Aware in the Standard Model, *EUROCRYPT'06*.
12. Y. Desmedt, Subliminal-Free Authentication and Signature (Extended Abstract), *EUROCRYPT 1988*, pp. 23-33, 1988.
13. M. Di Raimondo and R. Gennaro, New Approaches for Deniable Authentication, *ACM CCS'05*, pp. 112-121, 2005.
14. M. Di Raimondo, R. Gennaro and H. Krawczyk, Deniable Authentication and Key Exchange, *ACM CCS'06*.
15. D. Dolev, C. Dwork, M. Naor, Non-malleable Cryptography. *SIAM J. Comput.*, 30(2): 391-437 (2000). Earlier version appeared in *STOC'91*, pp. 542-552, 1991.
16. C. Dwork and M. Naor, Zaps and Their Applications, *FOCS'00*, pp. 283-293, 2000.
17. C. Dwork, M. Naor and A. Sahai, Concurrent Zero-Knowledge, *STOC'98*, pp. 409-418.
18. C. Dwork and A. Sahai, Concurrent Zero-Knowledge: Reducing the Need for Timing Constraints, *CRYPTO'98*, pp. 442-457.
19. U. Feige, A. Shamir, Zero Knowledge Proofs of Knowledge in Two Rounds, *Advanced in Cryptology-CRYPTO'89*, G. Brassard (Ed.), LNCS 435, Springer-Verlag, pp. 526-544, 1989.
20. S. Goldwasser, S. Micali, C. Rackoff, The Knowledge Complexity of Interactive Proof Systems, *SIAM J. Comput.*, 18(1): 186-208 (1989).
21. O. Goldreich, *Foundations of Cryptography: Basic Applications*, Cambridge University press, 2004.
22. J. Groth, R. Ostrovsky and A. Sahai, Perfect Non-interactive Zero Knowledge for NP, *EUROCRYPT'06*, pp. 339-358, 2006.
23. M. Jakobsson, K. Sako, R. Impagliazzo, Designated Verifier Proofs and Their Applications. *EUROCRYPT 1996*: 143-154.
24. S. Jiang and G. Gong, "Efficient Authenticators with Application to Key Exchange", *the 8th Annual International Conference on Information Security and Cryptology (ICISC'05)*, D. Won and S. Kim (Eds.), LNCS 3935, Springer-Verlag, pp. 81-91, 2006.
25. J. Katz, Efficient and Non-malleable Proofs of Plaintext Knowledge and Applications, *EUROCRYPT'03*, pp. 211-228.
26. J. Killian and E. Petrank, Concurrent and Resettable Zero-Knowledge in Poly-Logarithmic Rounds, *ACM STOC'01*, pp. 560-569, 2001.
27. H. Krawczyk, SKEME: a versatile secure key exchange mechanism for Internet, *NDSS'96*, pp. 114-127.
28. Y. Lindell, A Simpler Construction of CCA2-secure Public-key Encryption under General Assumptions, *EUROCRYPT'03*, pp. 241-254.
29. Y. Lindell, Lower Bounds and Impossibility Results for Concurrent Self Composition, to appear in *Journal of Cryptology*.

30. M. Naor, Deniable Ring Authentication, *Advances in Cryptology-CRYPTO'02*, M. Yung (Ed.), LNCS 2442, Springer-Verlag, pp. 481-498, 2002.
31. R. Pass, On the deniability in the common reference string and random oracle model, *Advances in Cryptology-CRYPTO'03*, D. Boneh (Ed.), LNCS 2729, Springer-Verlag, pp. 316-337, 2003.
32. M. Prabhakaran and A. Sahai, Concurrent Zero Knowledge Proofs with Logarithmic Round-Complexity, *FOCS'02*, pp. 366-375, 2002.
33. R. Richardson and J. Kilian, On the Concurrent Composition of Zero-Knowledge Proofs. *Advances in Cryptology-Eurocrypt'99*, pp. 415-431, 1999.
34. R. Rivest, A. Shamir and Y. Tauman, How to Leak a Secret, *ASIACRYPT'01*, pp. 552-565.

Appendix

Proof of Theorem 3. Keep the notations as in G_0 and G_1 . We need to construct \mathcal{A}_1 against the ideal process such that for any PPT machine \mathcal{Z} ,

$$\Pr[\mathcal{Z}(G_0) = 1] - \Pr[\mathcal{Z}(G_1) = 1] \quad (7)$$

is negligible. We still use the strategy outlined right before Theorem 2. We use \tilde{P}_i to denote party i in ideal process and P_i to denote party i in the execution of PKI-Auth simulated by \mathcal{A}_1 . Detailed code of \mathcal{A}_1 follows. Let \mathcal{C} be the set of parties corrupted by \mathcal{Z} . \mathbb{T} first initializes $\{(ek_i, dk_i), (sk_i, vk_i)\}_{i=1}^n$ and the common reference string crs of \mathcal{P} . Then it provides crs and $\{(dk_i, sk_i) : P_i \in \mathcal{C}\}$ as well as $\{ek_i, vk_i\}_{i=1}^n$ to \mathcal{A}_1 and \mathcal{Z} . Whenever \mathcal{Z} provides inputs (resp. instructions) to P_i (resp. \mathcal{A}_1), the following will be executed.

- I₁. Whenever \tilde{P}_i is given input $(\text{send}, \tilde{P}_j, m)$, he will copy to the input of $\hat{\mathcal{F}}$, who will further activate \mathcal{A}_1 with a message $(id, \tilde{P}_i, \tilde{P}_j, m)$. Then \mathcal{A}_1 simulates P_i to send a flow one message (m, a) to P_j and also reports this simulated message to \mathcal{Z} , where $a \leftarrow \{0, 1\}^\kappa$.
- I'₁ Whenever \mathcal{A}_1 is instructed, in the name of an corrupted P_i , to authenticate a message m to P_j , he sends $(\text{send}, \tilde{P}_j, m)$ to $\hat{\mathcal{F}}$. The remaining simulation of \mathcal{A}_1 in this query follows I₁.
- I₂. Whenever \mathcal{A}_1 is requested to activate P_j with a Flow1 message $m\|a$, \mathcal{A}_1 simulates P_j to execute as follows. Let P_i be the sender implied in m .
 - If $P_i \in \mathcal{C}$, then the action for P_j is normal, except that $RS_j(a, b, P_j, P_i, m, 0)$ is replaced by $RS_i(a, b, P_j, P_i, m, 0)$. Due to the property of ring signature, this simulation is perfect.
 - If $P_i \notin \mathcal{C}$, \mathcal{A} takes $b, k, u', u \leftarrow \{0, 1\}^\kappa$ and computes and sends out $m\|b\|E_i(k; u)\|E_j(k; u')\|\sigma$ to P_i , where $\sigma \leftarrow \mathcal{P}_{ij}\{E_i(k; u)\|E_j(k; u'); k\|u\|u'\}$. In addition, he adds a record

$$(a, b, P_j, P_i, m, 0, E_i(k; u)\|E_j(k; u'))$$

into a list \mathcal{L} which is initially empty.

In any case, report the Flow2 message to \mathcal{Z} .

- I₃. Whenever \mathcal{A}_1 is requested to activate P_i with a Flow2 message $m\|b\|c_1\|c_2\|\sigma$, \mathcal{A}_1 does the following.
 - If P_j (the receiver implied in m) is corrupted, then \mathcal{A}_1 uses dk_j to decrypt k from c_2 , verifies the validity of σ , and checks whether $RV_{ij}((a, b, P_j, P_i, 0), k) = 1$. If all the verifications are successful, define $z = RS_j(r, s, P_i, P_j, 1)$; otherwise, $z \leftarrow \{0, 1\}^\kappa$. The remaining simulation for a Flow3 message is normal.
 - If $P_j \notin \mathcal{C}$, \mathcal{A}_1 takes $v, z, v' \leftarrow \{0, 1\}^\kappa$, computes and sends out $m\|E_i(z; v)\|E_j(z; v')\|\omega$ to P_j , where $\omega \leftarrow \mathcal{P}_{ij}\{E_i(z; v)\|E_j(z; v'); z\|v\|v'\}$. In addition, if $(a, b, P_j, P_i, m, 0, c_1\|c_2) \in \mathcal{L}$ and $c_1\|c_2\|\sigma$ is consistent, \mathcal{A}_1 records

$$(a, b, P_i, P_j, m, 1, E_i(z; v)\|E_j(z; v'))$$

into \mathcal{L} .

In any case, \mathcal{A}_1 reports the Flow3 message to \mathcal{Z} .

- I₄. Whenever \mathcal{A}_1 is requested to activate P_j with a Flow3 message $m\|\gamma_1\|\gamma_2\|\omega$, \mathcal{A}_1 does the following. Let P_i be the sender implied in m .
- If $P_i \in \mathcal{C}$, then \mathcal{A}_1 decrypts z from γ_1 with dk_i . He then verifies whether $RV_{ij}((a, b, P_i, P_j, m, 1), z) = 1$ and whether ω is consistent with $\gamma_1\|\gamma_2$. He sets $\text{Flag}=1$ if both are verified successfully; otherwise, $\text{Flag} = 0$.
 - If $P_i \notin \mathcal{C}$, then \mathcal{A}_1 checks if $(a, b, P_i, P_j, m, 1, \gamma_1\|\gamma_2) \in \mathcal{L}$ and if $\gamma_1\|\gamma_2\|\omega$ is consistent, where $a\|b$ is taken from the internal state of P_j . In case of yes, he sets $\text{Flag}=1$; otherwise, $\text{Flag}=0$.
- In any case, if \exists a message $(id, \tilde{P}_i, \tilde{P}_j, m)$ was received from $\hat{\mathcal{F}}$ for some id but $(*, id)$ has not been feedback, he sends a message (Flag, id) back to $\hat{\mathcal{F}}$. If $\text{Flag} = 1$, simulate P_j to generate output $(\text{receiv}, id, P_i, P_j, m)$ where id is taken as in the (Flag, id) message (w.r.t. m) to $\hat{\mathcal{F}}$ if happens; $id \leftarrow \{0, 1\}^\kappa$ otherwise. Denote Bad_0 the event $\text{Flag} = 1$ but \mathcal{A}_1 never received $(*, \tilde{P}_i, \tilde{P}_j, m)$ from $\hat{\mathcal{F}}$. Note by item I₁, this happens only when P_i is uncorrupted. Denote Bad_1 the event P_i is uncorrupted and $\text{Flag} = 1$ but $(1, id)$ w.r.t. m was feedback to $\hat{\mathcal{F}}$ before. Note Bad_0 implies \tilde{P}_j will not output m but the simulated P_j does it; Bad_1 implies the simulated P_j outputs m twice but \tilde{P}_j only outputs it once.
- I₅. Whenever \mathcal{A} is requested to reveal a session state in P_t , he reports the partial or whole vector of (a, b, m, P_i, P_j) to \mathcal{Z} .

Denote Γ_0 the simulated execution of PKI-Auth by \mathcal{A}_1 . Consider a mental variant Γ_1 of Γ_0 , where the difference is that in the case of $P_j \notin \mathcal{C}$ at I₂ or I₃, all the records to \mathcal{L} are real (i.e., $k = RS_i(a, b, P_j, P_i, m, 0)$ for records from the former case and $w = RS_i(a, b, P_i, P_j, m, 1)$ for records from the latter case). Let Γ_2 denote the execution of PKI-Auth in the real world. Then Γ_2 and Γ_1 is identically distributed, except

- Case 1:** In the case $P_j \in \mathcal{C}$ at I₃ or $P_i \in \mathcal{C}$ at I₄, where the verifications in Γ_1 are successful but verifications fail in Γ_2 . This happens only when c_1 and c_2 at I₃ (or γ_1 and γ_2 at I₄) decrypts to different plaintexts. Denote this event of inconsistent ciphertexts in a game Γ_l by $\mathbf{E}_1(\Gamma_l), l = 0, 1, 2$. Then $\mathbf{E}_1(\Gamma_l)$ happens only if \mathcal{P}_{ij} in Γ_l has a soundness error. Let the runtime of \mathcal{Z} be bounded by R_z . Then $\Pr[\mathbf{E}_1(\Gamma_l)] \leq R_z \epsilon_1$.
- Case 2:** In case of $P_j \notin \mathcal{C}$ at I₃, $(a, b, P_j, P_i, m, 0, c_1, c_2) \notin \mathcal{L}$ but for $D_i(c_1) = k$ it holds that $V((a, b, P_j, P_i, m, 0), k) = 1$ and that $c_1|c_2|\sigma$ is consistent. Similarly, in case of $P_i \notin \mathcal{C}$ at I₄, $(a, b, P_i, P_j, m, 1, \gamma_1, \gamma_2) \notin \mathcal{L}$ but for $D_j(\gamma_2) = z$ it holds that $V_{vk}((a, b, P_i, P_j, m, 1), z) = 1$ and that $\gamma_1|\gamma_2|\omega$ is consistent. Denote either of these events in Game Γ_l ($l=0, 1$) by $\mathbf{E}_2(\Gamma_l)$. We have that in these events it must hold that $c_1|c_2$ (resp. $\gamma_1|\gamma_2$) encrypts an identical message k (resp. z); otherwise, one can reduce to break the soundness of \mathcal{P} . In the following, we always assume the soundness of \mathcal{P} holds. Upon an event $\mathbf{E}_2(\Gamma_l)$, if $(a, b, P_j, P_i, m, 0, \tilde{c}_1, \tilde{c}_2) \in \mathcal{L}$ in case of the former or $(a, b, P_i, P_j, m, 1, \tilde{\gamma}_1, \tilde{\gamma}_2) \in \mathcal{L}$ in case of the latter, then such an event happens with negligible probability; otherwise, one can reduce to break the non-malleability of E (Recall that E is CCA2 secure and that CCA2-security is equivalent to non-malleability under CCA2 attack; See [21, 15]). Details are omitted. We now consider the case $(a, b, P_j, P_i, m, 0, *, *) \notin \mathcal{L}$ (resp. $(a, b, P_i, P_j, m, 1, *, *) \notin \mathcal{L}$). Note that for any uncorrupted pair P_i and P_j , all the valid ring signatures in Game Γ_1 (or Γ_0 if any) generated by simulator must have been recorded in \mathcal{L} . Thus, unrecorded valid signature implies a forgery of ring signature for pair P_i and P_j . Denote the forgery of ring signature for an attacker with runtime R_z by ϵ_2 . Then we have $\Pr[\mathbf{E}_2(\Gamma_l)] \leq n^2 \epsilon_2$, for $l = 0, 1$.

Consider the gap between Γ_1 and Γ_0 . Since the only difference between them is whether records in \mathcal{L} are real or not, a simple Left-Right reduction for encryption allows us to reduce to the 2-user encryption left-right reduction¹, which in turn reduces to the semantical security of E in the multi-user setting (this again is equivalent to the semantic security of a single user; See Theorem 4.1 in [1]). Thus, the global outputs in Γ_1 and Γ_0 are indistinguishable.

Therefore, the global output of Γ_0 and Γ_2 are indistinguishable. To conclude the proof, we only need to guarantee that $\Pr[\mathbf{Bad}_0 \vee \mathbf{Bad}_1]$ is negligible. First of all, \mathbf{Bad}_0 does not happen since $\mathbf{Flag} = 1$ implies the sender session P_j was started. Recall the successful verification of the record at I_4 implies that P_i indeed processed Flow2 in order to send message m to P_j . By item I_1 , only after \tilde{P}_i sends $(\text{send}, \tilde{P}_j, m)$ to $\hat{\mathcal{F}}$, can \mathcal{A}_1 be invoked to simulate P_j (by the message $(id, *)$ from $\hat{\mathcal{F}}$). \mathbf{Bad}_1 occurs only if $\mathbf{Flag}=1$ at I_4 holds while \mathcal{A}_1 previously sent a message $(1, id)$ to $\hat{\mathcal{F}}$. This event implies that b repeats in P_j . Thus, it occurs with probability at most $\frac{R_z^2}{2^\kappa}$, negligible. This concludes the proof. \blacksquare

¹ Toward this reduction, we need to run the simulator of \mathcal{P} . We remark that the NIZK simulator here is only used to prove the gap between Γ_1 and Γ_0 is negligible. Our real simulator \mathcal{A}_1 in Γ_0 does not use this simulator thus does not break the deniability restriction. The game simulated by the left-right attacker is indistinguishable from Γ_0 (resp. Γ_1); otherwise, zero knowledge property of \mathcal{P} is broken.