Security Analysis of Standard Authentication and Key Agreement Protocols Utilising Timestamps

M. Barbosa and P. Farshim

Departamento de Informática, Universidade do Minho,
Campus de Gualtar, 4710-057 Braga, Portugal.
{mbb,farshim}@di.uminho.pt

Abstract. We propose a generic modelling technique that can be used to extend existing frameworks for theoretical security analysis in order to capture the use of timestamps. We apply this technique to two of the most popular models adopted in literature (Bellare-Rogaway and Canetti-Krawczyk). We analyse previous results obtained using these models in light of the proposed extensions, and demonstrate their application to a new class of protocols. In the timed CK model we concentrate on modular design and analysis of protocols, and propose a more efficient timed authenticator relying on timestamps. The structure of this new authenticator implies that an authentication mechanism standardised in ISO-9798 is secure. Finally, we use our timed extension to the BR model to establish the security of an efficient ISO protocol for key transport and unilateral entity authentication.

Keywords. Timestamp, Key Agreement, Entity Authentication.

1 Introduction

The analysis of key agreement protocols has received a lot of attention within the cryptographic community, as they are central components in secure communication systems. Theoretical treatment of these protocols has been performed under computational models of security, under symbolic models and, more recently, under hybrid models which bridge the gap between these two approaches. However, a common trait to all previous work in this area is the abstraction of time, even when key agreement protocols are explicitly synchronous and resort to representations of local time in their definitions. The use of timestamps in key distribution protocols was suggested by Denning and Sacco [13]. Nowadays, protocols such as Kerberos [19], the entity authentication protocols in ISO-9798 [16,17], and the key agreement protocols in ISO-11770 [15] rely on timestamps. In this paper we are concerned with the formal security analysis of such protocols.

Perhaps the most common use of timestamps in cryptographic protocols is to counteract replay and interleaving attacks, and to provide uniqueness or timeliness guarantees [18, Section 10.3.1]. In this sense, timestamps are an alternative to challenge-response mechanisms using fresh random nonces and to message sequence numbers. The typical operation of a system relying on timestamps is the following. Before sending a message, a party obtains a time reading from its local clock, and cryptographically binds it to the message for transmission. The receiver of the message obtains the current time from its own local clock, and subtracts the value of the timestamp received. The message is immediately rejected if the timestamp difference is outside a fixed-size time interval, called an *acceptance window*, selected to account for the maximum message transit and processing time, plus clock skew between the two parties. Furthermore, the receiver keeps a list of all previously received messages whose timestamps are still within the acceptance window. The incoming message is also checked against this list, and it is rejected if it is found to be a replay.

The fact that the receiver must keep local state to detect the replay of valid timestamped messages within the acceptance window may also be considered a disadvantage. For example, a solution based on a challenge-response mechanism using a random nonce could be used for the same purpose and would only require the receiver to keep a small amount of short-term state (albeit per-connection). In comparison to challenge-response mechanisms, protocols using timestamps will typically require one less message to complete and will not require parties to generate random numbers. On the downside, the receiver must keep a small amount of ephemeral local state to detect the replay of valid messages within an acceptance window. The amount of state that must be kept when using timestamps can also be seen as an advantage when compared, for example, with solutions using sequence numbers where the receiver must keep static long-term state for each possible peer. In other application scenarios, there is no real alternative to the use of timestamps. Examples of this are the implementation of time-limited privileges, such as those awarded by Kerberos tickets, or the legal validity of authenticated documents, such as X.509 public key certificates.

In short, timestamps are extensively used in cryptographic protocols and they are adopted in mainstream (de facto) cryptographic standards, because they have interesting security properties that can be advantageous in many real-world scenarios. However, to the best of our knowledge, the use of timestamps has not been addressed in previously published work on the theoretical security analysis of cryptographic protocols. In particular, the current formal security models for the analysis of cryptographic protocols do not allow capturing this sort of mechanism in any reasonable way.

The security of this sort of mechanism relies on the use of a common time reference. This means that each party must have a local clock and that these must be synchronised to an extent that accommodates the acceptance window that is used. The local clocks must also be secure to prevent adversarial modification: if an adversary is able to reset a clock backwards, then it might be able to restore the validity of old messages; conversely, by setting a clock forward, the adversary might have advantage in preparing a message for some future point in time. These assumptions on the security and synchronisation of local clocks may be seen as disadvantages of using timestamps, since in many environments they may not be realistic. For example, it is common that the synchronisation of local clocks in a distributed environment is enforced by communication protocols that must themselves be secure in order for this assumption to be valid.

Our contribution. In this paper, we propose a general approach to the enrichment of said models to permit analysing protocols relying on timestamps. Our focus is on a generic modelling technique, which can be applied to virtually any framework for the analysis of cryptographic protocols. For concreteness, we apply this technique to two of the most popular models adopted in literature (the family of models stemming from the work of Bellare and Rogaway [4] and the model proposed by Canetti and Krawczyk in [2]), analyse previous results obtained using these models in light of the proposed extensions, and demonstrate their application to a new class of protocols.

An additional contribution of this paper is that the examples we use to demonstrate our approach are standardised protocols that lacked a formal proof of security until now. In particular, the timestamped authenticator we present in Section 4 was described in a footnote in the original paper by Canetti and Krawczyk in [2], but no proof of security was provided to support the claim. Furthermore, the structure of this new authenticator (and the security proof we provide) imply that a signature-based unilateral authentication mechanism standardised in ISO-9798-3 is secure for message authentication. Similarly, to the best of our knowledge, the ISO-11770-2 key transport protocol we analyse in Section 5 previously lacked a formal proof of security to validate the informal security guarantees described in the standard.

Structure of the paper. In Section 2 we briefly review the related work. In Section 3 we introduce our modelling approach and then use it propose extensions to the BR model and the CK model that permit capturing timestamping techniques, and discuss the implications for previous results. Finally, we present two examples of how the extended models can be used to analyse concrete protocols: an efficient authenticator in the CK model in Section 4, and a one-pass key exchange protocol from ISO-11770-2 in Section 5. We conclude the paper with a discussion on directions for future work in Section 6.

2 Related work

Bellare and Rogaway [4] gave the first formal model of security for the analysis of authentication and key agreement protocols. It is a game-based definition, in which the adversary is allowed to interact with a set of oracles that model communicating parties in a network, and where the adversary's goal is to distinguish whether the challenge it is given is a correctly shared key or is a randomly generated value. This seminal paper also provided the first computational proof of security for a cryptographic protocol. Subsequent work by the same authors [5] corrected a flaw in the original formulation and a considerable number of publications since have brought the model to maturity. This evolution included the simplification and refinement of the concept of matching conversations, the introduction of session identifiers and the observation that these are most naturally defined as message traces, the adaptation of the model to different scenarios, and the use of the model to capture different security goals [6]. In the remainder of this paper we will refer to the security models that follow this approach of Bellare and Rogaway as BR models.

In [2] Bellare, Canetti and Krawczyk proposed a modular approach to the design and analysis of authentication and key agreement protocols. This work adapted the concept of simulatability, and showed how one could analyse the security of a protocol in an ideally authenticated world (the authenticated model, where the adversary must stick to legitimate message delivery) and then use an authenticator to compile it into a new protocol providing the same security guarantees in a more realistic model. (the unauthenticated model, where the adversary is able to alter and inject messages arbitrarily). Canetti and Krawczyk [11] later corrected some problems with the original formulation of this model by merging their simulation-based approach (in particular they maintained the notions of emulation and compilation that enable the modular construction of protocols) with an indistinguishability-based security definition for key exchange protocols. This enabled the authors to prove a composition theorem, whereby combining a key agreement protocol with an authenticated encryption scheme for message transfer yields a two-stage secure message exchange system. In the remainder of this paper we will refer to the security models that follow this approach of Canetti and Krawczyk as CK models.

This line of work subsequently led to Canetti's proposal of the powerful and general Universal Composability framework [10] for the analysis of cryptographic protocols, and to the further development of the study of key exchange security models in the UC paradigm in [12]. The Universal Composability framework is based on the simulatability paradigm, whereby a protocol is proven to realise a specified ideal (secure) functionality by means of a simulator that interacts with the ideal functionality and is able to emulate the behaviour of any adversary attacking the protocol. A proof of security in the UC framework provides guarantees that security holds even if the protocol is run in parallel with other copies of itself, and even with multiple instances of other protocols. A disadvantage of the UC framework is that it imposes notions of security which are often too strong to capture practical schemes. However, a relaxed version of this framework is shown to be equivalent to the CK model in [12].

Handling of time in related work. Cryptographic protocols are analysed in abstract models, where participants and adversaries are represented by processes exchanging messages through communication channels. Central to these models is the way they capture the timeliness of physical communication networks. In particular, it is possible to split these models into two categories by looking at the way they handle the activation of processes and the delivery of sent messages. In synchronous models, time is captured as a sequence of rounds. In each round all processes are activated simultaneously, and messages are exchanged instantly. In asynchronous models, there is no explicit assumption on the global passing of time. Process activation is usually message-driven and the adversary controls message delivery and participant activation.

Synchronous models are usually adapted when the focus is on a timeliness guarantee, such as termination of a process. However, asynchronous models are taken as better abstraction of real communication systems, as they make no assumptions about network delays and the relative execution speed of the parties, and they nicely capture the view that communications networks are hostile environments controlled by malicious agents [1]. For this reason, asynchronous models, such as the ones described earlier in this section, are much more widely used. This trend, however, comes at the cost of abstracting away many of the practical uses of time-variant parameters in cryptographic protocols, which rely on explicit representations of time. For example, it is common practice to treat timestamps as random nonces, or to assume that all transmitted messages are different. This is an understandable strategy to simplify analyses, but misses security-critical protocol implementation aspects such as buffering previously received messages to avoid replay attacks, or the use of timestamps and windows of acceptance to reduce the size of said message buffers [18, Section 10.3.1].

3 Adding time awareness to BR and CK models

3.1 General approach

The objective is to obtain a framework for the analysis of key agreement protocols relying on timestamps, where one can argue that they satisfy a formal security definition. We do not introduce an all-new time-aware analysis framework, which would mean our findings might break away from the current state-of-the-art and might not be easily comparable to previously published results. Instead, we propose to extend the existing models for the analysis of key agreement protocols in a natural way, taking care to preserve an acceptable degree of backward-compatibility. The basic idea of our approach is applicable to several competing analysis frameworks that are currently used by researchers in this area, and it does not imply the adoption of any particular one.

To demonstrate this principle, we propose to extend the BR and CK models referred in Section 2 in very similar terms. The most important change that we introduce is that we provide the communicating parties with internal clocks. These clocks are the only means available to each party to determine the current (local) time. To preserve the common asynchronous trait in these models, where the adversary controls the entire sequence of events occurring during an execution, we do not allow the clocks to progress independently. Instead, we leave it to the adversary to control the individual clocks of parties: we allow it to perform a `Tick` (or activation) query through which it can increment the internal clock of an honest party (of course it has complete control of the clocks of corrupted parties). The adversary is not allowed to reset or cause the internal clocks to regress in any way. This restriction captures the real-world assumption we described in Section 1 that the internal clocks of honest parties must be, to some extent, secure.

The addition of these elements to the BR and CK models allows us to capture the notion of time and internal clock drifts. We preserve the asynchronous nature of the model by allowing the adversary to freely control the perception of time passing at the different parties. Through the `Tick` mechanism, the adversary is able to induce any conceivable pattern in the relative speed of local clocks, and may try to use this capability to obtain advantage in attacking protocols that rely on local time measurements to construct and/or validate timestamps. Of course by giving this power to the adversary, we are enabling it to drive internal clocks significantly out of synchrony with respect to each other. However, a secure protocol using explicit representations of time should make it infeasible for an adversary to take advantage of such a strategy, or at least should permit formally stating the amount of drift that can tolerated. At this point, it is important to distinguish two types of security guarantees that may be obtained from timestamps and that we aim to capture using this modelling strategy.

Resistance against replay attacks. Recall that, in protocols that use timestamps to prevent replay attacks, the receiver defines an acceptance window and temporarily stores received messages until their timestamps expire. The width of the acceptance window must be defined as a trade-off between the required amount of storage space, the expected message transmission frequency, speed and processing time; and the required synchronisation between the clocks of sender and receiver. Received messages are discarded if they have invalid timestamps, or if they are repeats within the acceptance window.

In this setting, the local clocks are not explicitly used to keep track of elapsed time, but simply to ensure that the receiver does not have to store all previously received messages to prevent accepting duplicates. In fact, for this purpose, timestamps are essentially equivalent to sequence numbers. Furthermore, synchronisation of clocks between sender and receiver is less of a timeliness issue, and more of an interoperability problem. For example, two honest parties using this mechanism might not be able to communicate at all, even without the active intervention of any adversary, should their clocks values be sufficiently apart. In our extended model, this is reminiscent of a Denial-of-Service attack, which is usually out of the scope of cryptographic security analyses. Consistently with this view and with the original models, the security definitions for cryptographic protocols using timestamps in this context remain unchanged: it is accepted that the adversary may be able to prevent successful completions of protocols (e.g. by driving internal clocks significantly out of synchronisation, or simply by not delivering messages) but it should not be able to break the security requirements in any useful way.

Timeliness guarantees. For protocols that use timestamps to obtain timeliness guarantees on messages, the local clock values are taken for what they really mean: time measurements. In this context, timestamped messages are typically valid for a longer period of time, and timeliness guarantees can be provided to either the sender or the receiver, or to both. For example, the sender may want to be sure that a message will not be accepted by an honest receiver outside its validity period, which is defined with respect to the sender's own internal clock. Conversely, the receiver may require assurance that an accepted message was generated *recently* with respect to its own local clock, where *recently* is quantifiable as a time interval.

To deal with these guarantees we need to capture accuracy assumptions on the internal clocks of the honest parties in the system. We can do this by imposing limits on the maximum pair-wise drift that the adversary can induce between the internal clocks of different parties. In our modelling approach, we capture this sort of security requirement by stating that a protocol enforcing such a timeliness property must guarantee that any adversary breaking this requirement must be overstepping its maximum drift allowance with overwhelming probability.

3.2 Extending the CK model

Brief review of the CK model [2,11]. An n-party message-driven protocol is a collection of n programs. Each program is run by a different party with some initial input that includes the party's identity, random input and the security parameter. The program waits for an *activation*: (1) the arrival of an *incoming message* from the network, or (2) an *action request* coming from other programs run by the party. Upon activation, the program processes the incoming data, starting from its current internal state, and as a result it can generate outgoing messages to the network and action requests to other programs run by the party. In addition, a *local output* value is generated and appended to a cumulative output tape, which is initially empty. The protocol definition includes an *initialisation* function I that models an initial phase of out-of-band and authenticated information exchange between the parties. Function I takes a random input r and the security parameter κ, and outputs a vector $I(r,\kappa) = I(r,\kappa)_0, \ldots, I(r,\kappa)_n$. The component $I(r,\kappa)_0$ is the public information that becomes known to all parties and to the adversary. For $i > 0$, $I(r,\kappa)_i$ becomes known only to P_i.

The *Unauthenticated-Links Adversarial Model* (UM) defines the capabilities of an active man-in-the-middle attacker and its interaction with a protocol [11]. The participants are parties $P_1, \ldots, P_n$ running an n-party protocol π on inputs $x_1, \ldots, x_n$, respectively, and an adversary $\mathcal{U}$. For initialisation, each party P_i invokes π on local input x_i, security parameter κ and random input; the initialisation function of π is executed as described above. Then, while $\mathcal{U}$ has not terminated do:

1. $\mathcal{U}$ may `activate` π within some party, P_i. An activation can take two forms:
   (a) An `action request` q. This activation models requests or invocations coming from other programs run by the party.
   (b) An `incoming message` m with a specified sender P_j. This activation models messages coming from the network. We assume that every message specifies the sender of the message and its intended recipient.

   If an activation occurred then the activated party P_i runs its program and hands $\mathcal{U}$ the resulting outgoing messages and action requests. We stress that $\mathcal{U}$ is free to choose any scheduling of activations and determine the values of incoming messages. In particular, incoming messages need not correspond in any way to messages that have been sent. That is, $\mathcal{U}$ is free to generate, inject, modify, and deliver any message of its choice. Local outputs produced by the protocol are known to $\mathcal{U}$ except for those labeled `secret`.
2. $\mathcal{U}$ may `corrupt` a party P_i. Upon corruption, $\mathcal{U}$ learns the current internal state of P_i, and a special message is added to P_i's local output. From this point on, P_i is no longer activated and does not generate further local output. Note, in particular, that upon corruption of P_i the attacker learns $I(r,\kappa)_i$.
3. $\mathcal{U}$ may issue a `session-state reveal` for a specified session within some party P_i. In this case, $\mathcal{U}$ learns the current internal state of the specified session within P_i. This event is recorded through a special note in P_i's local output.
4. $\mathcal{U}$ may issue a `session-output query` for a specified session within some party P_i. In this case, $\mathcal{U}$ learns any output from the specified session that was labeled `secret`. This event is recorded through a special note in P_i's local output.

The *global output* of running a protocol in the UM is the concatenation of the cumulative local outputs of all the parties, together with the output of the adversary. The global output resulting from adversary $\mathcal{U}$ interacting with parties running protocol π is seen as an ensemble of probability distributions parameterised by security parameter $k \in \mathbb{N}$ and the input to the system[1] $\mathbf{x} \in \{0,1\}^*$, and where the probability space is defined by the combined coin tosses of the adversary and the communicating parties. Following the original notation, we denote this ensemble by $\mathtt{UNAUTH}_{\mathcal{U},\pi}$.

The *Authenticated-Links Adversarial Model* (AM) is identical to the UM, with the exception that the adversary is constrained to model an ideal authenticated communications system. The AM-adversary, denoted $\mathcal{A}$ cannot inject or modify messages, except if the specified sender is a corrupted party or if the message belongs to an exposed session. Formally, when a message is sent by a party, the message is added to a set M of *undelivered messages*. Whenever $\mathcal{A}$ activates a party P_j on some incoming message m it must be that m is in the set M and that P_j is the intended recipient in m. Furthermore, m is now deleted from M. Note that $\mathcal{A}$ is not required to maintain the order of the messages, nor is it bound by any fairness requirement on the

[1] The concatenation of global public data with individual local inputs for each party.

activation of parties, nor is it required to deliver all messages. There are also no limitations on the *external requests* that $\mathcal{A}$ issues. Analogously to $\texttt{UNAUTH}_{\mathcal{U},\pi}$, we have that $\texttt{AUTH}_{\mathcal{A},\pi}$ is the ensemble of random variables representing the global output for a computation carried out in the authenticated-links model.

Due to space limitations, we present the definitions related to Key-Exchange protocols and their security in the CK model in Appendix A.

Introducing local clocks. Our modification to the previous model is based on a special program that we call `LocalTime`.

Definition 1 (`LocalTime` Program). *The* `LocalTime` *program follows the syntax of message-driven protocols. The program does not accept messages from the network or transmit messages to the network. The program is deterministic and it is invoked with the empty input. It maintains a* `clock` *variable as internal state, which is initialised to* 0*. The program accepts a single external request, with no parameters, which is called* `Tick`*. When activated by the* `Tick` *request, the program increments the counter and outputs* `Local Time: <clock>` *, where* `<clock>` *denotes the value of the* `clock` *variable.*

We introduce the *timed* variants of the UM and AM, which we refer to as TUM and TAM, and we require that each party in the TUM and in the TAM runs a single instance of `LocalTime`. Note that in the timed models, the adversary may control the value of the internal `clock` variables at will, by sending the `Tick` request to any party. Consistently with the original models, we assume that the local output at a given party P_i is readable only by the adversary and the programs running in the same party. Alternatively, the internal `clock` variable can be seen as part of the local state of each party, which is read-only to other programs and protocols running in the same environment. This means, in particular, that a program which enables a party P_i to participate in a given protocol may use the local clock value at that party, but is otherwise unaware of any other time references. We disallow protocols from issuing the `Tick` request to their local clock themselves. [2].

Remark. The approach we followed to integrate the local clocks into the communicating parties in the CK model deserves a few words of explanation. Firstly, the adversary's interactions with parties in the CK model are either external requests to protocols, or message deliveries. Our choice of modelling the local clock as a separate program that accepts the `Tick` activation as an external request is consistent with this principle, and allows the adversary to control the local clock as desired. Secondly, by causing the `LocalTime` program to produce local output after each tick, protocol outputs do not need to include information about the time at which a certain event occurred in order to make timeliness properties explicit: this follows directly from the cumulative nature of the local output. Finally, our approach makes the concept of protocol emulation time-aware: the fact that the local clock progression is observable in the local output of each party also implies that any protocol π' that emulates a protocol π (see Definition 2 below) is guaranteed to preserve any timeliness properties formulated over the global output when the original protocol is run in the TAM.

Modular protocol design in the timed models. Central to the methodology of [2] are the concepts of *protocol emulation*, *compiler*, and *authenticator*, which we directly adapt for protocol translations between the TAM and the TUM.

Definition 2 (Emulation). *A protocol π' emulates a protocol π in the TUM if, for any TUM adversary $\mathcal{U}_T$, there exists TAM adversary $\mathcal{A}_T$ such that, for all input vectors, the global output resulting from running $\mathcal{A}_T$ against π in the TAM is computationally indistinguishable from that obtained when running $\mathcal{U}_T$ against π' in the TUM.*

We emphasise that the global outputs resulting from running a protocol in the timed models include the local outputs produced by the `LocalTime` program, which reflect the sequence of `Tick` queries performed by the adversary at each party, and that these outputs are captured by the emulation definition above.

Definition 3 (Timed-Authenticator). *A* compiler *$\mathcal{C}$ is an algorithm that takes for input descriptions of protocols and outputs descriptions of protocols. A* timed-authenticator *is a compiler $\mathcal{C}$ that, for any TAM protocol π, the protocol $\mathcal{C}(\pi)$ emulates π in the TUM.*

[2] We leave this possibility for future work, as well as the more interesting variant in which a party can run a protocol to keep its internal clock synchronised according to some criteria. We may also empower the adversary by allowing him access to an interface which sets the local time at a party to an arbitrary provided value. Although in this paper we only deal with a `Tick`ing adversary, we mention that if reasonable restrictions on clock behaviour, such as its drift, are imposed, analogous security proofs to those in this paper may be obtained.

We also state the following theorem, which is equivalent to Theorem 6 in [11]. We omit the proof, as it is almost identical to the original one.

Theorem 1. *Let π be a SK-secure (See Appendix A) key-exchange protocol in the TAM and let $\mathcal{C}$ be a timed-authenticator. Then $\pi' = \mathcal{C}(\pi)$ is a SK-secure key-exchange protocol in the TUM.*

In Section 4 we prove that, not only the original AM-to-UM authenticators proposed in [2] are also timed-authenticators, but also that through the use of timestamps one can obtain more efficient timed-authenticators. However, in order to argue that these results are meaningful, we need to revisit the modular approach to the development of cryptographic protocols introduced in [2]. With the introduction of the timed models, we have now four options for the design and analysis of protocols. For convenience, one would like to carry out the design in the authenticated models (AM and TAM), where adversaries are more limited in their capabilities and security goals are easier to achieve. The choice of whether or not to use a timed model should depend only on whether or not the protocol relies on time-dependent parameters to achieve security. On the other hand, and without loss of generality, we will assume that the overall goal is to translate these protocols into the TUM, which is the most general of the more realistic unauthenticated models, given that it accommodates protocols which may or may not take advantage of the local clock feature. To support this methodology, we first formalise a class of protocols for which the timed models are not particularly relevant.

Definition 4 (Time-Independence). *A protocol π is* time-independent *if its behaviour is oblivious of the* `LocalTime` *protocol, i.e. if protocol π does not use the outputs of the* `LocalTime` *protocol in any way.*

One would expect that, for time-independent protocols, the TUM (resp. TAM) would be identical to the UM (resp. AM). In particular, all of the results obtained in [11] for specific time-independent protocols should carry across to the timed models we have introduced. Unfortunately, proving a general theorem establishing that, for any time-independent protocol, in the UM (resp. AM) one can simply recast it in the TUM (resp. TAM) to obtain a protocol which emulates the original one (and satisfying the same security definitions) is not possible given our definition of the `LocalTime` program. This is because it is, by definition, impossible to recreate local time outputs by individual parties in the UM (resp. AM), and hence a simulation-based proof does not go through. However, for the specific case of SK-security, we can prove the following theorem establishing that, for time-independent protocols, one can perform the analysis in the UM (resp. AM) and the results will still apply in the TUM (resp. TAM).

Theorem 2. *If a time-independent UM-protocol (resp. AM-protocol) π is SK-secure, then it is also SK-secure when run in the TUM (resp. TAM).*

Remark. We emphasise that, although we are able to show that the newly proposed timed models are a coherent extension to the work in [2,11] for the design and analysis of key exchange protocols relying on time-dependent parameters, we are not able to establish a general theorem that carries through all of the previous results in the CK model. In particular, we cannot prove a theorem stating that AM-to-UM emulation implies TAM-to-TUM emulation for time-independent protocols. This would automatically imply that all authenticators are also timed-authenticators (we will return to this discussion in Section 4). However, the proof for such a theorem does not seem to go through because the definition of emulation is not strong enough to guarantee that, using the existence of suitable AM adversary for all TUM-adversaries, one is able to construct the required TAM-adversary that produces an indistinguishable sequence of `Tick` queries.

Theorem 2, combined with the concrete time-dependant and time-independent timed-authenticators in Section 4, provides the desired degree of flexibility in designing SK-secure KE protocols, as shown in the table below.

Lower Layer	Time-independent authenticator	Time-dependent authenticator
Time-independent in the AM	Use the original CK modular approach to obtain an SK-secure protocol in the UM. Apply Theorem 2 to move to the TUM.	Use Theorem 2 to move result to the TAM. Apply the timed-authenticator in Section 4 to obtain an SK-secure KE protocol in the TUM.
Time-dependent in the TAM	Apply one of the original authenticators in [2], which are also timed-authenticators by Theorem 3, to obtain an SK-secure KE in the TUM.	Apply the timed-authenticator in Section 4 to obtain an SK-secure KE protocol in the TUM.

3.3 Extending the BR model

Brief review of the BR model [4,6]. Protocol participants are the elements of a non-empty set $\mathcal{ID}$ of principals. Each principal $A \in \mathcal{ID}$ is named by a fixed-length string, and they all hold public information and long-lived cryptographic private keys. Everybody's private key and public information is determined by running a key generator. During the execution of a protocol, there may be many running instances of each principal $A \in \mathcal{ID}$. We call instance i of principal A an oracle, and we denote it Π_A^i. Each instance of a principal might be embodied as a process (running on some machine) which is controlled by that principal.

Intuitively, protocol execution proceeds as follows. An initiator-instance speaks first, producing a first message. A responder-instance may reply with a message of its own, intended for the initiator-instance. This process is intended to continue for some fixed number of flows, until both instances have *terminated*, by which time each instance should also have *accepted*. Acceptance may occur at any time, and it means that the party holds a session key sk, a session identifier sid (that can be used to uniquely name the ensuing session), and a partner identifier pid (that names the principal with which the instance believes it has just exchanged a key). The session key is secret, but the other two parameters are considered public. An instance can accept at most once.

Adversary $\mathcal{A}$ is defined as a probabilistic algorithm which has access to an arbitrary number of instance oracles, as described above, to which he can place the following queries:

– $\texttt{Send}(A, B, i, m)$: This delivers message m, which is claimed to originate in party B, to oracle Π_A^i. The oracle computes what the protocol says to, and returns back the response. Should the oracle accept, this fact, as well as the session and partner identifiers will be made visible to the adversary. Should the oracle terminate, this too will be made visible to the adversary. To initiate the protocol with an instance A posing as initiator, and an instance of B as responder, the adversary should call $\texttt{Send}(A, B, i, \lambda)$ on an unused instance i of A.
– $\texttt{Reveal}(A, i)$: If oracle Π_A^i has accepted, holding some session key sk, then this query returns sk to the adversary.
– $\texttt{Corrupt}(A)$: This oracle returns the private key corresponding to party A[3].
– $\texttt{Test}(A, i)$: If oracle Π_A^i has terminated, holding some session key sk and $pid = B$, then the following happens. A coin b is flipped. If $b = 1$, then sk is returned to the adversary. Otherwise, a random session key, drawn from the appropriate distribution, is returned.

To capture the security of authenticated key agreement protocols (AKE), we require the following definitions.

Definition 5 (Partnering). *We say that Π_A^i is the partner of $\Pi_{A'}^{i'}$ if (1) Both oracles has accepted and hold (sk, sid, pid) and (sk', sid', pid') respectively; (2) $sk = sk'$, $sid = sid'$, $pid = A'$, and $pid' = A$; and (3) No oracle besides Π_A^i and $\Pi_{A'}^{i'}$ has accepted with session identity sid. Note that partnership is symmetric.*

Definition 6 (Freshness). *Π_A^i is fresh if no reveal or corrupt queries are placed on Π_A^i or its partner Π_B^j.*

Definition 7 (AKE Security). *We say that a key exchange protocol is AKE secure if for any probabilistic polynomial-time adversary $\mathcal{A}$, the probability that $\mathcal{A}$ guesses the bit b chosen in a fresh test session is negligibly different from $1/2$. The advantage of the adversary, which returns a bit b', is defined to be:*

$$\mathrm{Adv}_{\mathtt{KE}}^{\mathtt{AKE}}(\mathcal{A}) := |2\Pr[b = b'] - 1|.$$

Definition 8 (Entity Authentication (EA)). *We say that a key exchange protocol provides initiator-to-responder authentication if, for any probabilistic polynomial-time adversary $\mathcal{A}$ attacking the protocol in the above model, the probability that some honest responder oracle Π_B^j terminates with $pid = A$, an honest party, but has no partner oracle is negligible. We denote this probability by $\mathrm{Adv}_{\mathtt{KE}}^{\mathtt{I2R}}(\mathcal{A})$.*

Remark. The restriction of being honest that we have imposed above, is introduced to model the setting where authentication relies on symmetric keys. This is the case for the protocol we analyse in Section 5. In the asymmetric setting, however, only the authenticated party (initiator in the above) needs to be honest.

[3] For simplicity we adopt the weak corruption model, where $\texttt{Corrupt}$ does not return the states of all instances of A.

Introducing local clocks. To ensure consistency with the structure of the BR model, we provide each party with a `clock` variable, which is initially set to zero. This variable is read-only state, which is accessible to all the instances of a protocol running at a given party (very much like the private keys). In order to model the adversarial control of clocks at different parties we enhance its capabilities by providing access to the following oracle:

– `Tick`(A): This oracle increments the `clock` variable at party A, and returns its new value.

It is interesting to note that the relation between the timed version of the BR model and the original one is identical to that we established in the previous section between the TUM and the UM in the CK model. Specifically, one can formulate the notions of AKE security and entity authentication without change in the timed BR model. It is also straightforward to adapt the definition of time-independence to protocols specified in the BR model and prove that, for all time-independent protocols, AKE security and entity authentication are preserved when we move from the original to the timed version of the model. We omit the equivalent of Theorem 2 for BR models due to space limitations. However, the observation that such a theorem holds is important to support our claim that the extension we propose to the BR model is a natural one.

Capturing timeliness guarantees. The definition of entity authentication formulated over the timed BR model is a good case study for capturing timeliness guarantees in security definitions. The existential guarantee stated in the definition implicitly refers to two events: (1) the termination of the protocol-instance that obtains the authentication guarantee; and (2) the acceptance of the partner protocol-instance that is authenticated. It seems natural to extend this definition with additional information relating the points in time at which the two events occur.

To achieve this, we must first isolate a category of adversaries for which making such claims is possible.

Definition 9 (δ-synchronisation). *An adversary in the timed BR model satisfies δ-synchronisation if it never causes the* `clock` *variables of any two (honest) parties to differ by more than δ.*

The previous definition captures the notion that clocks must be synchronised in order to achieve any sort of timeliness guarantee, as described in Section 3.1. We are now in a position to state an alternative version of the entity authentication definition. Let Π^i_A and Π^j_B be two partner oracles where the latter has terminated. Also, let $t_B(E)$ be the function returning the value of the local clock at B when event E occurred. Finally, let $\texttt{acc}(A, i)$ denote the event that Π^i_A accepted, and let $\texttt{term}(B, j)$ denote the event that Π^j_B terminated.

Definition 10 (β-Recent Entity Authentication (β-REA)). *We say that a key exchange protocol provides β-recent initiator-to-responder authentication if it provides initiator-to-responder authentication, and furthermore for any honest responder oracle Π^j_B which has terminated with partner Π^i_A, with A honest, we have that: $|t_B(\texttt{term}(B, j)) - t_B(\texttt{acc}(A, i))| \le \beta$.*

The above definition captures attacks such as that described in [14], where an adversary uses a post-dated clock at a client to impersonate as him later, when correct time is reached at the server side. In Section 5 we will prove that a concrete key agreement protocol using timestamps satisfies the previous definition, as long as the adversary is guaranteed to comply with δ-synchronisation.

4 An example in the CK model: timed-authenticators

The concept of *authenticator* is central to the modular approach to the analysis of cryptographic protocols proposed in [2]. Authenticators are compilers that take protocols shown to satisfy a set of properties in the AM, and produce protocols which satisfy equivalent properties in the UM. Bellare et al. [2] propose a method to construct authenticators based on the simple message transfer (MT) protocol: they show that any protocol which emulates the MT protocol in the UM can be used as an authenticator. Authenticators constructed using in this way are called *MT-authenticators.*

In this section we show that this method can be easily adapted to the timed versions of the CK model introduced in the previous section. We start by recalling the definition of the MT-protocol and note that, when run in the timed models, the local output at each party permits reading the local time at which the MT-protocol signalled the reception and transmission of messages.

Definition 11 (The MT-Protocol). *The protocol takes empty input. Upon activation within P_i on action request* `Send`(P_i, P_j, m)*, party P_i sends the message (P_i, P_j, m) to party P_j, and outputs "P_i* `sent` *m* `to` *P_j". Upon receipt of a message (P_i, Pj, m), P_j outputs "P_j* `received` *m* `from` *P_i".*

Now, let λ be a protocol that emulates the MT-protocol in the TUM and, similarly to the modular construction in [2], define a compiler $\mathcal{C}_\lambda$ that on input a protocol π produces a protocol $\pi' = \mathcal{C}_\lambda(\pi)$ defined as follows.

– When π' is activated at a party P_i it first invokes λ.
– Then, for each message sent in protocol π, protocol π' activates λ with the action request for sending the same message to the same specified recipient.
– Whenever π' is activated with some incoming message, it activates λ with the same incoming message.
– When λ outputs "P_i `received` m `from` P_j", protocol π is activated with incoming message m from P_j.

We complete this discussion with two theorems. Theorem 3 is the equivalent of Theorem 3 in [2]. Theorem 4 is the equivalent of Propositions 4 and 5 in [2].

Theorem 3. *Let π be a protocol in the TAM, and let λ be protocol which emulates the MT-protocol in the TUM, then $\pi' = \mathcal{C}_\lambda(\pi)$ emulates π in the TUM.*

Theorem 4. *The signature-based and the encryption-based MT-authenticators proposed in [2] both emulate the MT-protocol in the TUM.*

The proofs are identical to the original ones, with the following exception: when a TUM-adversary activates the `LocalTime` protocol of a (dummy) TUM-party by a `Tick` request, the simulating TAM-adversary invokes the `LocalTime` protocol of the corresponding party in the TAM, and passes back the output, without change, to the TUM-adversary.

Theorem 4 establishes that the original compilers proposed in [2] can also be used to translate protocols from the TAM to the TUM, i.e. they are also timed-authenticators. Intuitively, this is possible because these constructions are oblivious of time and of the `LocalTime` programs added to the timed models, and the MT-protocol is not claimed to provide concrete timeliness guarantees.

Protocol $\lambda_{Sig}(\delta)$

– The initialisation function I first invokes, once for each party, the key generation algorithm of a signature scheme secure against chosen message attacks with security parameter κ. Let `Sig` and `Ver` denote the signing and verification algorithms. Let s_i and v_i denote the signing and verification keys associated with party P_i. The public information includes all public keys: $I_0 = v_1, \ldots, v_n$. P_i's private information is $I_i = s_i$.
– Each party keeps as protocol state a list L where it stores message/timestamp pairs (m, t) corresponding to previously received and accepted messages.
– When activated within party P_i and with external request to send message m to party P_j, protocol $\lambda_{Sig}(\delta)$ invokes a two-party protocol that proceeds as follows:
  - First, P_i checks the local time value t and constructs a message $(m||t||\texttt{Sig}(m||t||P_j, s_i))$ and sends it to P_j.
  - Then, P_i outputs "P_i `sent` m `to` P_j".
  - Upon receipt of $(m||t||\sigma)$ from P_i, party P_j accepts m and outputs "P_j `received` m `from` P_i" if:
    * the signature σ is successfully verified by $\texttt{Ver}(m||t||P_j, v_i)$.
    * $t \in [t' - \delta, t' + \delta]$, where t' is the value of the local time at P_j when the message is received.
    * list L does not contain the pair (m, t).
  - Finally, P_j updates list L adding the pair (m, t) and deleting all pairs $(\hat{m}, \hat{t})$ where $\hat{t} \notin [t' - \delta, t' + \delta]$.

Fig. 1. A signature-based timed-authenticator in the TUM

To complete this section, we present a more efficient one-round timed authenticator, which uses timestamps to eliminate the challenge-response construction used in the original authenticators. The protocol is shown in Figure 1, and is an adaptation of the signature-based MT-authenticator in [2]. It is parameterised with a positive integer δ which defines the width of the timestamp acceptance window. We observe that this protocol is structurally equivalent to the signature-based unilateral authentication protocol in the ISO-9798-3 standard [17] when one uses message m in place of the optional text-fields allowed by the standard. This implies that Theorem 5 below establishes the validity of the claim in standard ISO-9798-3 that this protocol can be used for message authentication.

The following theorem formally establishes the security properties of the protocol in Figure 1. The proof of this theorem can be found in Appendix B.

Theorem 5. *Assume that the signature scheme in use is secure against the standard notion of chosen message attacks (UF-CMA). Then protocol $\lambda_{Sig}(\delta)$ emulates the MT-protocol in the TUM.*

Remark. There is a subtlety involving adversaries who can forge signatures but do not disrupt the simulation needed by a TAM-adversary. Consider an adversary who activates party P^* to send message m at local time t^*, but does not deliver the message to the intended recipient. Instead, it forges a signature on the same message, but with a later timestamp, and delivers this message to the intended recipient much later in the simulation run, taking care that the timestamp in the forged message is valid at that time. This adversary does not cause a problem in the proof of the above theorem, since the message is delivered only once. In fact, this is an attack on the timeliness properties of the authenticator, which are not captured in the formulation of the MT-protocol. This attack would be an important part of the proof that protocol $\lambda_{Sig}(\delta)$ emulates a version of the MT-protocol with timeliness guarantees, where messages are only accepted on the receiver's side if they are delivered within some specific time interval after they are added to the set M.

5 An example in the BR model: a standard AKE protocol

In this section we use the timed BR model to analyse the security of a one-pass key agreement protocol offering unilateral authentication, as defined in the ISO-11770-2 standard. The protocol is formalised in Figure 2. It is a key transport protocol that uses an authenticated symmetric encryption scheme (see Appendix C) to carry a fresh session key between the initiator and the responder. The use of timestamps permits achieving AKE security in one-pass, and the reception of the single message in the protocol effectively allows the responder to authenticate the initiator. In fact, this protocol is presented in the ISO-11770-2 standard as a particular use of a unilateral authentication protocol presented in ISO-9798-2, where the session key is transmitted in place of a generic text field. As explained in the remark in Section 3.3, the security proof we present here can be easily adapted to show that the underlying ISO-9798-2 protocol is a secure unilateral EA protocol.

ISO-11770-2 informally states the following security properties for the protocol in Figure 2. The session key is supplied by the initiator party, and AKE security is guaranteed by the confidentiality property of the underlying authenticated encryption scheme. The protocol provides unilateral authentication: the mechanism enables the responder to authenticate the initiator. Entity authentication is achieved by demonstrating knowledge of a secret authentication key, i.e. the entity using its secret key to encipher specific data. For this reason, the protocol requires an authenticated encryption algorithm which provides, not only data confidentiality, but also data integrity and data origin authentication. Uniqueness and timeliness is controlled by timestamps: the protocol uses timestamps to prevent valid messages (authentication information) from being accepted at a later time or more than once.

The protocol requires that parties are able to maintain mechanisms for generating or verifying the validity of timestamps: the deciphered data includes a timestamp that must be validated by the recipient. Parties maintains a list L to detect replay attacks. In relation to forward secrecy, note that if an adversary gets hold of a ciphertext stored in L, and furthermore at some point it corrupts the owner of the list, it can compute the secret key for the corresponding past session. Identifier B is included in the ciphertext to prevent a substitution attack, i.e. the re-use of this message by an adversary masquerading as B to A. In environments where such attacks cannot occur, the identifier may be omitted [15].

The following theorem formally establishes the security properties of the protocol in Figure 2. The proof of this theorem may be found in Appendix D.

Theorem 6. *The protocol $\pi_{\texttt{AuthEnc}}(\delta)$ in Figure 2 is an AKE secure key exchange protocol in the timed BR model if the underlying authenticated encryption scheme is secure in the IND-CPA and INT-CTXT senses (see Appendix C). This protocol also provides initiator-to-responder authentication if the authenticated encryption scheme is INT-CTXT secure. More precisely we have:*

$$\mathrm{Adv}^{\texttt{I2R}}_{\texttt{KE}}(\mathcal{A}) \leq 2q^2 \cdot \mathrm{Adv}^{\texttt{INT-CTXT}}_{\texttt{AuthEnc}}(\mathcal{B}_1) + q^2 q_s/|\mathcal{K}|,$$

Protocol $\pi_{\texttt{AuthEnc}}(\delta)$

- The initialisation function I first invokes, once for each pair of parties, the key generation algorithm of an authenticated symmetric encryption scheme with security parameter κ and sets the secret information of party A with pair B to be $K_{A,B}$. I_A is set to be the list of the keys A shares with B for all parties B.
- All parties keep as protocol state a list L where it stores ciphertext/timestamp pairs (c, t) corresponding to previously received and accepted messages.
- When activated within party A to act as initiator, and establish a session with party B, the protocol proceeds as follows.
  - A checks the local time value t. It generates a random session key sk and constructs the ciphertext $c \leftarrow \texttt{AuthEnc}(sk||t||B, K_{A,B})$. It then sends (A, B, c) to B.
  - A accepts sk as the session key, (A, B, c) as sid, B as pid, and terminates.
- Upon receipt of (A, B, c), the responder accepts a key sk as the session key, (A, B, c) as sid, A as pid and terminates if:
  - B is the identity of responder.
  - c successfully decrypts to $(sk||t||B)$ under $K_{A,B}$.
  - $t \in [t' - \delta, t' + \delta]$, where t' is the value of the local time at B when the message is received.
  - List L does not contain the pair (c, t).
- Finally, B updates the list L, adding the pair (c, t) and deleting all pairs $(\hat{c}, \hat{t})$ where $\hat{t} \notin [t' - \delta, t' + \delta]$.

Fig. 2. One-pass key agreement with unilateral authentication from ISO-11770-2

$$\mathrm{Adv}^{\texttt{AKE}}_{\texttt{KE}}(\mathcal{A}) \leq q^2(2 + q_s) \cdot \mathrm{Adv}^{\texttt{INT-CTXT}}_{\texttt{AuthEnc}}(\mathcal{B}_1) + q^2 q_s \cdot \mathrm{Adv}^{\texttt{IND-CPA}}_{\texttt{AuthEnc}}(\mathcal{B}_2) + q^2 q_s/|\mathcal{K}|.$$

Here a uniform distribution on the key space $\mathcal{K}$ is assumed, q is the maximum number of parties involved in the attack, and q_s is the maximum number of sessions held at any party.

Furthermore, if the adversary respects β-synchronisation, then the protocol guarantees $(\beta + \delta)$-recent initiator-to-responder authentication.

6 Conclusion

In this paper we proposed a general modelling technique that can be used to extend current models for the analysis of key agreement protocols, so that they capture the use of timestamps. We have shown that two popular analysis frameworks (CK and BR models) can be extended in a natural way using this technique, and that this permits addressing a new class of real-world protocols that, until now, lacked a complete formal treatment. The paper also leaves many open problems that can be addressed in future work. We conclude the paper by referring some of these topics. The approach we introduced can be applied to extend other theoretical models, the most interesting of which is perhaps the Universal Composability framework of Canetti [10]. Orthogonally, there are many key agreement and authentication protocols which rely on timestamps and that could benefit from a security analysis in a time-aware framework. Kerberos [19] is an example of such a protocol, which utilises timestamps in a setting where a server is available. In order to rigourously analyse the security of this protocol, one would need to define a timed version of three-party key agreement security models. Moving away from key agreement and authentication protocols, our approach opens the way for the formal analysis of time-related cryptographic protocols such as those aiming to provide secure message timestamping and clock-synchronisation. Finally, it would be interesting to see how one could apply a similar approach to security models that try to capture public key infrastructures, where the temporal validity of certificates is usually ignored.

Acknowledgments

The authors would like to thank Alex Dent for proposing and discussing the original ideas that led to this work. They would also like to thank Bogdan Warinschi for important discussions. The authors were funded in part by the WITS project (FCT - PTDC/EIA/71362/2006) and the eCrypt II project (EU FP7 - ICT-2007-216646). The second author was also supported in part by the Scientific and Technological Research Council of Turkey (TÜBİTAK) while at Middle East Technical University.

References

1. M. Backes. Unifying Simulatability Definitions in Cryptographic Systems under Different Timing Assumptions. In *International Conference on Concurrency Theory (CONCUR)*, LNCS 2761:350-365, Springer-Verlag, 2003.
2. M. Bellare, R. Canetti and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 419–428, ACM Press, 1998.
3. M. Bellare and C. Namprempre. Authenticated Encryption: Relations Among Notions and Analysis of the Generic Composition Paradigm. In *Advances in Cryptology – ASIACRYPT 2000*, LNCS 1976:531–545, Springer-Verlag, 2000.
4. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *Advances in Cryptology – CRYPTO 93*, LNCS 773:232–249, Springer-Verlag, 1994.
5. M. Bellare and P. Rogaway. Provably Secure Session Key Distribution: The Three Party Case. In *Proceedings of the 27th Annual Symposium on the Theory of Computing*, pages 57–66, ACM Press, 1995.
6. M. Bellare, D. Pointcheval and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In *Advances in Cryptology – EUROCRYPT 2000*, LNCS 1807:139–155, Springer-Verlag, 2000.
7. B. Blanchet, A.D. Jaggard, A. Scedrov and J.-K. Tsay. Computationally Sound Mechanised Proofs for Basic and Public-Key Kerberos. In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security*, pages 87–99, ACM Press, 2008.
8. B. Blanchet and D. Pointcheval. Automated Security Proofs with Sequences of Games. In *Advances in Cryptology – CRYPTO 2006*, LNCS 4117:537–554, Springer-Verlag, 2006.
9. A. Boldyreva and V. Kumar. Provable-Security Analysis of Authenticated Encryption in Kerberos. In *2007 IEEE Symposium on Security and Privacy*, pages 92–100, IEEE Computer Society, 2007.
10. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, IEEE Computer Society, 2001.
11. R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *Advances in Cryptology – EUROCRYPT 2001*, LNCS 2045:453–474, Springer-Verlag, 2001.
12. R. Canetti and H. Krawczyk. Universally Composable Notions of Key-Exchange and Secure Channels. In *Advances in Cryptology – EUROCRYPT 2002*, LNCS 2332:337–351, Springer-Verlag, 2002.
13. D.E. Denning and G.M. Sacco. Timestamps in Key Distribution Protocols. In *Communications of the ACM*, Volume 24, Issue 8, Pages: 533–536, ACM, 1981.
14. L. Gong. A Security Risk of Depending on Synchronized Clocks. In *ACM SIGOPS Operating Systems Review*, Volume 26, Issue 1, Pages: 49–53. ACM, 1992.
15. ISO/IEC 11770-2: 2008, Information Technology – Security Techniques – Key Management – Part 2: Mechanisms Using Symmetric Techniques.
16. ISO/IEC 9798-2: 1999, Information Technology – Security Techniques – Entity Authentication – Part 2: Mechanisms Using Symmetric Encipherment Algorithms.
17. ISO/IEC 9798-3: 1998, Information Technology – Security Techniques – Entity Authentication – Part 3: Mechanisms Using Digital Signature Techniques.
18. A.J. Menezes, P.C. van Oorschot and S.A. Vanstone. *Handbook of Applied Cryptography*, CRC Press, 2001.
19. C. Newman, T. Yu, S. Hartman and K. Raeburn. The Kerberos Network Authentication Service (V5). `http://www.ietf.org/rfc/rfc4120`, 2005.

Appendix A – Key Exchange protocols in the CK model

Key-exchange[4] (KE) protocols are n-party message-driven protocols where the program within each party, P_i, takes action requests of the form $\texttt{establish- session}$ $(P_i, P_j, \texttt{sid}, \texttt{role})$. Here P_j is another party, $\texttt{sid}$ is a session identifier, and $\texttt{role} \in \{\texttt{initiator}, \texttt{responder}\}$. Local outputs of a KE protocol are of the form $(P_i, P_j, \texttt{sid}, k)$, where k is a session key. A null value of k is interpreted as a *session abortion* and will usually represent the termination of the session with a returned error message. Non-null session-key values are labelled secret. Each session is aimed at exchanging a single key with a specified party. The protocol ensures that a session is never created within P_i with repeat inputs $(P_i, P_j, \texttt{sid})$. It is assumed by convention that, once a session returns, its entire local state is deleted. If in an execution of a KE protocol P_i has a session with input $(P_i, P_j, \texttt{sid}, \texttt{role})$ and party P_j has a session with input $(P_j, P_i, \texttt{sid}', \texttt{role}')$, and $\texttt{sid} = \texttt{sid}'$ then we say that the two sessions are *matching* and we call P_i and P_j the *partners* of session $\texttt{sid}$. A session

[4] For simplicity, we will omit the concepts of session expiration and Perfect Forward Secrecy (PFS) from our presentation, but we note that they can be adapted without change to the timed models described in this paper.

$(P_i, P_j, \mathtt{sid})$ is called *locally exposed* within P_i, if the attacker performed any of the following actions on said session: (i) a session-state reveal; (ii) a session-key query; (iii) corruption of P_i. A session is called *exposed* if it is locally exposed or it has a matching session that is locally exposed. A session which is not exposed is called *unexposed.*

To define the security of a KE protocol in the UM (resp. AM), we extend the usual capabilities of the adversary $\mathcal{U}$ (resp. $\mathcal{A}$) by allowing it to perform a `test-session` query: we let the adversary choose, at any time during its run, a test-session among the sessions that are completed and unexposed at the time. Let k be the value of the corresponding session-key. We toss a coin b and, if $b = 1$ we provide the adversary with the value k. Otherwise we provide it with a value randomly chosen from the same space as k. The attacker is now allowed to continue with the regular actions of a UM (resp. AM) adversary but is not allowed to expose the test-session. At the end of its run, the adversary outputs a bit b' (as its guess for b).

Definition 12 (SK-Security). *A KE protocol π is called SK-secure in the UM (resp. AM) if the following properties hold for any KE adversary $\mathcal{U}$ (resp. $\mathcal{A}$) in the UM (resp. AM).*

1. *Protocol π satisfies the property that if two honest parties complete matching sessions then they both output the same key; and*
2. *The probability that $\mathcal{U}$ (resp. $\mathcal{A}$) guesses correctly the bit b is no more than $1/2$ plus a negligible fraction in the security parameter.*

Appendix B – Proof of Theorem 5

Proof. Let $\mathcal{U}$ be a TUM-adversary that interacts with $\lambda_{Sig}(\delta)$. We construct an TAM-adversary $\mathcal{A}$ such that the global outputs induced by the two adversaries in the corresponding adversarial models are statistically close. $\mathcal{A}$ runs $\mathcal{U}$ on a simulated interaction with a set of parties running $\lambda_{Sig}(\delta)$. First $\mathcal{A}$ chooses and distributes keys for the imitated parties, according to function I. Next, $\mathcal{A}$ executes the simulated interaction between $\mathcal{U}$ and the imitated parties running $\lambda_{Sig}(\delta)$, taking action in the TAM when the following events occur:

– When $\mathcal{U}$ activates some imitated party P'_i with a `Tick` request, then $\mathcal{A}$ performs a `Tick` request on the corresponding TAM party P_i.
– When $\mathcal{U}$ activates some imitated party P'_i for sending a message m to imitated party P'_j, adversary $\mathcal{A}$ activates party P_i in the TAM network to send m to P_j.
– When some imitated party P'_j outputs "P'_j `received` $\hat{m}$ `from` P'_i", adversary $\mathcal{A}$ activates party P_j in the TAM model with incoming message $\hat{m}$ from $\mathcal{A}$.
– When $\mathcal{U}$ corrupts a party, $\mathcal{A}$ corrupts the same party in the authenticated network and hands the corresponding information (from the simulated run) to $\mathcal{U}$.
– When $\mathcal{U}$ terminates, $\mathcal{A}$ outputs whatever $\mathcal{U}$ outputs.

Let E denote the event that imitated party P'_j outputs "P'_j `received` m `from` P'_i" where P'_i is honest and the message (m, P_i, P_j) is not currently in the set M of undelivered messages. In other words, E is the event where P_j should receive message m to ensure a perfect simulation, but either P_i wasn't activated for sending m to P_j, or P_j has already had the same output before (we are assuming that no message is transmitted twice). In this event we say that $\mathcal{U}$ *broke* party P'_i, since the rules of an ideal authenticated network captured by the TAM do not allow $\mathcal{U}$ to deliver a message which was not sent by P_i. It is straightforward to see that if event E does not occur, then the simulation run by $\mathcal{A}$ is perfect and the global outputs in the TUM and in the TAM are identically distributed.

It remains to show that event E occurs only with negligible probability. Assume that event E occurs with probability ϵ. We construct a forger $\mathcal{F}$ that breaks the signature scheme with probability ϵ/n. Given a verification key v^* and access to a signing oracle, forger $\mathcal{F}$ runs $\mathcal{U}$ on the following simulated interaction with a set of parties running $\lambda_{Sig}(\delta)$ in the TUM. First $\mathcal{F}$ chooses and distributes keys for the imitated parties according to function I, with the exception that the public verification key associated with some party P^*, chosen at random, is replaced with the input key v^*. Next, $\mathcal{F}$ executes the simulated interaction taking advantage of its knowledge of public keys and all-but-one secret keys and using the following exceptional steps when required.

– If during the simulation P^* is required to sign a value l then $\mathcal{F}$ asks its signing oracle for a signature of l.

- If party P^* is corrupted then the simulation is aborted and $\mathcal{F}$ fails.
- If an event equivalent to E occurs (a message is accepted by a party P_j which would not be allowed in the TAM), then $\mathcal{F}$ announces success and outputs a forgery as described below.

First note that $\mathcal{U}$'s view of the interaction with $\mathcal{F}$, conditioned on the event that $\mathcal{F}$ does not abort the simulation, is identically distributed to $\mathcal{U}$'s view of a real interaction with a TUM network running $\lambda_{Sig}(\delta)$ (this is so since P^* is randomly chosen). Let E^* be the event where an event equivalent to E occurs in the simulated run of $\mathcal{U}$ within $\mathcal{F}$, and that the party broken by $\mathcal{F}$ is P^*. Since P^* is chosen at random, and since event E^* and an abortion never occur at the same run, we have that event E^* occurs with probability at least ϵ/n.

Assume that event E^* occurs, and that party P_j triggered it by accepting message m with signature $(m||t||P_j)$. We have two cases.

- In case that P^* was not activated to send m to P_j, it is clear that P^* never signed a string $(m||t||P_j)$ and the signature in the last received message is a valid forgery. This is the value output by $\mathcal{U}$.
- In the case that P_j outputs the same value twice, we again assume without loss of generality that all transmitted messages are different and we have that P^* sent m only once with timestamp t^*. Additionally, we know that P_j would not accept two messages with the same pair (m, t^*) due to the list L. Hence, we have two cases:
  - Suppose that $t = t^*$. Then it must be the case that P_j previously received and accepted a message with a valid signature on a string with a different (earlier) timestamp. Since P^* never signed this message, then this is the string that $\mathcal{U}$ returns as a valid forgery.
  - Similarly, if $t \neq t^*$ then P^* never signed a string $(m||t||P_j)$ and the signature in the last received message is a valid forgery, which $\mathcal{U}$ returns.

From this discussion it follows that $\mathcal{F}$ has successfully broken the signature scheme with non-negligible probability, contradicting the security assumption on the signature scheme. □

Appendix C – Authenticated Symmetric Encryption

An authenticated symmetric encryption scheme [3] is defined via a triple of probabilistic polynomial-time algorithms as follows.

1. $\texttt{Gen}_{\texttt{AuthEnc}}(1^\kappa)$: This algorithm on input a security parameter 1^κ outputs a key K.
2. $\texttt{Enc}_{\texttt{AuthEnc}}(m, K)$: This algorithm on input a message m and a key K returns a ciphertext c.
3. $\texttt{Dec}_{\texttt{AuthEnc}}(c, K)$: This algorithm is deterministic and on input a ciphertext c and a key K returns a message m or an error symbol $\perp$.

The INT-CTXT security definition shown below captures the integrity of the messages encrypted under the scheme.

```
INT-CTXT
 1. $K \leftarrow \texttt{Gen}_{\texttt{AuthEnc}}(1^\kappa)$
 2. $c \leftarrow \mathcal{A}^{\mathcal{O}}(1^\kappa)$
```

$$\mathrm{Adv}^{\texttt{INT-CTXT}}_{\texttt{AuthEnc}}(\mathcal{A}) := \Pr[\texttt{Dec}_{\texttt{AuthEnc}}(c, K) \neq \perp \text{ and } c \notin S].$$

In the above $\mathcal{O}$ denotes an encryption oracle which on input a message m returns $\texttt{Enc}_{\texttt{AuthEnc}}(m, K)$. Here S denotes the set of ciphertexts obtained from the encryption oracle.

We may add a decryption oracle to the above to obtain a new game which we call INT-CXT$'$. This game if related to the original game via:

$$\mathrm{Adv}^{\texttt{INT-CTXT}'}_{\texttt{AuthEnc}}(\mathcal{A}) \leq 2 \cdot \mathrm{Adv}^{\texttt{INT-CTXT}}_{\texttt{AuthEnc}}(\mathcal{B}).$$

To see this, note that we may simulate a decryption oracle in the INT-CTXT game by always returning $\perp$ on the ciphertexts not obtained from the encryption oracle. Let E be the event that the adversary queries this simulated oracle on a valid ciphertext. The probability of this event is exactly the probability of winning the INT-CTXT game. The claim follows by the difference lemma:

Lemma 1 (Difference Lemma). *Let W_1, W_2 and E be events in the same probability space for which $W_1 \wedge \neg E = W_2 \wedge \neg E$. Then*

$$|\Pr[W_1] - \Pr[W_2]| \leq \Pr[E].$$

The confidentiality of the encrypted messages is modelled via the following IND-CCA definition of security.

```
IND-CPA
 1. $K \leftarrow \mathtt{Gen}_{\mathtt{AuthEnc}}(1^\kappa)$
 2. $b \leftarrow \{0, 1\}$
 3. $b' \leftarrow \mathcal{A}^{\mathcal{O}}(1^\kappa)$
```

$$\mathrm{Adv}_{\mathtt{AuthEnc}}^{\mathtt{IND-CPA}}(\mathcal{A}) := |2\Pr[b' = b] - 1|.$$

Here $\mathcal{O}$ denotes a left-right encryption oracle which on input a message pair (m_0, m_1) returns $\mathtt{Enc}_{\mathtt{AuthEnc}}(m_b, K)$.

Again we may give the adversary access to a decryption oracle, which cannot be called on ciphertext returned from the encryption oracle except those which were the result of calling the left-right encryption oracle on two identical messages. This results in the IND-CCA game.

INT-CTXT together with IND-CPA security implies security in the IND-CCA sense [3]:

$$\mathrm{Adv}_{\mathtt{AuthEnc}}^{\mathtt{IND-CCA}}(\mathcal{A}) \leq \mathrm{Adv}_{\mathtt{AuthEnc}}^{\mathtt{INT-CTXT}}(\mathcal{B}_1) + \mathrm{Adv}_{\mathtt{AuthEnc}}^{\mathtt{IND-CPA}}(\mathcal{B}_2).$$

To see this, note that similarly to the argument given for the INT-CTXT′ game, we may simulate a decryption oracle by simply returning $\perp$ on any query except those resulted from calling the left-right oracle with equal messages (in this case we return the message).

Appendix D – Proof of Theorem 6

Proof. Let us first reformulate Definition 8 to an equivalent version which is easier to work with. This new definition, which we call I2R′, is identical to the original, with the difference that the adversary places an additional (dummy) `Test` query on a terminated honest responder oracle Π_B^j at the end of its run. Its advantage, $\mathrm{Adv}_{\mathtt{KE}}^{\mathtt{I2R}'}(\mathcal{A})$, is the probability that the `pid` in the test session indicates a party which is honest, and yet no partner initiator instance exists at that party. We note that, if an adversary can force some responder oracle to terminate without a partner, it can actually pinpoint this oracle, as this is an observable event. Therefore, we have

$$\mathrm{Adv}_{\mathtt{KE}}^{\mathtt{I2R}}(\mathcal{A}) = \mathrm{Adv}_{\mathtt{KE}}^{\mathtt{I2R}'}(\mathcal{A}).$$

Now let $\mathcal{A}$ be an adversary that tries to win the I2R′ game against the protocol in the timed BR model. We show, via a sequence of modified attack games, that the advantage of such an adversary is negligible. We do so by first guessing the identities that will be involved in the dummy `Test` query placed by $\mathcal{A}$, and then reduce the advantage that it may have in violating initiator-to-responder entity authentication, to a successful attack in the INT-CTXT′ game against the underlying authenticated encryption scheme (see Appendix B for definition and relation to INT-CTXT game).

$\mathtt{Game}_0$: This game is identical to the original security game:

$$\mathrm{Adv}_{\mathtt{KE}}^{\mathtt{Game}_0}(\mathcal{A}) = \mathrm{Adv}_{\mathtt{KE}}^{\mathtt{I2R}'}(\mathcal{A}).$$

$\mathtt{Game}_1$: This game attempts to guess the identities who will act as the initiator and the responder in the test session. To this end, it chooses two indexes $i_A, i_B \in \{1, \ldots, q\}$. It defines A^* and B^* to be the identities of the parties corresponding to these indices respectively (which will become known as during the game). If the dummy test query is not placed on an oracle who has accepted, as responder, with $sid = (A^*, B^*, c^*)$, for some ciphertext c^*, this game declares the adversary a loser and terminates. It is straightforward to check that:

$$\mathrm{Adv}_{\mathtt{KE}}^{\mathtt{Game}_1}(\mathcal{A}) = 1/q^2 \cdot \mathrm{Adv}_{\mathtt{KE}}^{\mathtt{Game}_1}(\mathcal{A}).$$

$\mathtt{Game}_2$: This game declares the adversary a loser if:

– Event E: oracle $\Pi^i_{B^*}$, the responder where the dummy test query is placed, terminated with $sid^* = (A^*, B^*, c^*)$, but none of A^*'s instances terminated with this session identifier.

By the difference lemma we have:

$$|\mathrm{Adv}^{\mathtt{Game}_1}_{\mathtt{KE}}(\mathcal{A}) - \mathrm{Adv}^{\mathtt{Game}_2}_{\mathtt{KE}}(\mathcal{A})| \leq \Pr[E].$$

We show that there is an adversary $\mathcal{B}_1$ which successfully forges a ciphertext in the INT-CTXT$'$ security game if event E occurs. More precisely we show

$$\Pr[E] \leq \mathrm{Adv}^{\mathtt{INT-CTXT}'}_{\mathtt{AuthEnc}}(\mathcal{B}_1).$$

Algorithm $\mathcal{B}_1$ operates as follows.

– It chooses two indexes $i_A, i_B \in \{1, \ldots, q\}$. It defines A^* and B^* to be the identities of the parties corresponding to these indexes respectively.
– At the initialisation stage, it generates keys for all (unordered) pairs of participants according to the key distribution, except for the pair $\{A^*, B^*\}$.
– It simulates various oracles as follows.
– $\mathtt{Send}(A, B, i, m)$: algorithm $\mathcal{B}_1$ proceeds as follows.
  - If the pair $\{A, B\}$ is different from $\{A^*, B^*\}$, then $\mathcal{B}_1$ knows $K_{A,B}$ and it proceeds as follows.
    * If $m = \lambda$, then A is the initiator, and it encrypts $(sk||t_A||B)$ for a random key sk under $K_{A,B}$. Here t_A is obtained by reading the current value of the $\mathtt{Tick}$ oracle corresponding to A.
    * Otherwise, it completes A's reception of the message using the key $K_{A,B}$, its current $\mathtt{Tick}$ value, and the list L. It processes the list L according to protocol rules (appends L with the ciphertext/timestamp pair, deleting expired entries).
  - If the pair involved is $\{A^*, B^*\}$, then algorithm $\mathcal{B}_1$ does not know K_{A^*,B^*}, and it operates as follows.
    * If $m = \lambda$, then A is the initiator, and it uses the encryption oracle on $(sk||t_A||B)$, for a random key sk.
    * Otherwise, it completes A's reception of the message using the decryption oracle, A's current $\mathtt{Tick}$ value and the list L. It processes the list L according to protocol rules.

  In all of the above cases, the values of the secret keys sk are stored to answer $\mathtt{Reveal}$ queries later on.
– $\mathtt{Corrupt}(A)$: algorithm $\mathcal{B}_1$ returns the keys it generated in the initialisation stage. If $A \in \{A^*, B^*\}$, algorithm $\mathcal{B}_1$ terminates with $\perp$. This termination is consistent with $\mathtt{Game}_2$, as the parties involved in the dummy test session cannot be corrupt if the adversary is to break the authentication property.
– $\mathtt{Reveal}(A, i)$: algorithm $\mathcal{B}_1$ returns the secret key sk that it previously stored for that session, as a result of a $\mathtt{Send}$ query.
– $\mathtt{Tick}(A)$: these are answered by maintaining a counter for each party. Note that $\mathcal{B}_1$ has access to the current value of each $\mathtt{Tick}$ oracle.
– $\mathtt{Test}(A, i)$: algorithm $\mathcal{B}_1$ follows the normal procedure for test queries: it tosses a coin and, depending on its value, passes to the adversary the legitimate key or a randomly generated value in the key space.

When the adversary terminates, algorithm $\mathcal{B}_1$ identifies the (responder) instance of B^* at which the adversary placed its dummy test query, and returns the value of the ciphertext c^* that caused it to accept and terminate. Note that $\mathcal{B}_1$'s simulation is perfect according to the rules of $\mathtt{Game}_1$ and $\mathtt{Game}_2$. Furthermore, as described above, the adversary's advantage in both games is the same unless event E occurred. We show that if event E occurred, then c^* is a valid forgery.

When this ciphertext is returned, the identities in the dummy test session are A^* and B^*, where the former is the initiator and the latter is the responder. Since ciphertext c^* is accepted (the dummy test session must accept), we know that c^* is a valid ciphertext with respect to the (unknown) K_{A^*,B^*}. It remains to check that the encryption oracle never returned c^*. This oracle is called only when a message is sent from A^* to B^*, or vice-versa. However, since event E occurred, then c^* was never transmitted by A^*. Hence, c^* could only have been returned from this oracle as a result of placing a $\mathtt{Send}(B^*, A^*, i, \lambda)$ query. Suppose c^* came from such a query. According to the protocol definition, c^* will be rejected if the responder's identity, B^*, does not match that under c^*. This means that $A^* = B^*$ and the query was of the form $\mathtt{Send}(A^*, A^*, i, \lambda)$, which again contradicts event E. This implies that c^* is a valid forgery, and that if event E occurs with non-negligible probability, then $\mathcal{B}_1$ succeeds with the same advantage in winning the INT-CTXT$'$ game.

We have shown that for any honest responder instance at B^*, which has terminated with $sid^* = (A^*, B^*, c^*)$, its intended (honest) initiator instance at A^*, has also accepted sid^* with overwhelming probability. By unique decryptability of ciphertexts and symmetry of private keys, these oracles hold the same sk as well. However, this is still not sufficient to prove that the protocol provides initiator-to-responder authentication. In fact, the property requires that, not only the two instances have accepted with the same session identifier, but also that no other oracle has accepted with the same identifier. We now argue that this is also true with overwhelming probability.

To show this, we start by observing that only oracles instances of A^* and B^* could possibly accept with the same session identifier, given that this includes the identities of the participating parties (no instance of a third party with a different identity could possibly accept with the same session identifier). We are now in a position to introduce a final game, where no adversary can possibly win, i.e. the rules of the game make it impossible to break the initiator to responder entity authentication property.

$\mathtt{Game}_3$: This game declares the adversary a loser if there exist a third oracle which accepts with sid^* .

We show, for the uniform distribution on the key space $\mathcal{K}$, that:

$$|\mathrm{Adv}_{\mathtt{KE}}^{\mathtt{Game}_2}(\mathcal{A}) - \mathrm{Adv}_{\mathtt{KE}}^{\mathtt{Game}_3}(\mathcal{A})| \leq q_s/|\mathcal{K}|.$$

Suppose that three or more oracles have accepted $sid^* = (A^*, B^*, c^*)$, the session identifier of the dummy test session. Since any oracle is either an initiator or a responder, and the identities corresponding to these roles are fixed in the session identifier, we have that either two initiator oracles with identity A^*, or two responder oracles with identity B^* have accepted with sid^*. The probability that any initiator A^* accepts a repeat sid^* is at most $q_s/|\mathcal{K}|$: sk is chosen uniformly randomly and independently in each of the q_s possible instances. Furthermore, a responder instance at B^* will never accept sid^* repeatedly: it will always recover the same timestamp t from c^* (it decrypts using $K_{A,B}$ in both sessions), and then either its local time will be outside the window of acceptance around t, or it is still within and a replay will be detected.

This concludes the proof that the protocol provides initiator-to-responder authentication. We now extend the above to establish AKE security. For this, we again introduce a sequence of games. $\mathtt{Game}_0$ now denotes the AKE security game. In $\mathtt{Game}_1$ we simply assume that the adversary has zero advantage in violating the initiator-to-responder entity authentication property of the protocol: should this property be violated, the adversary is immediately declared a loser. This gives us:

$$|\mathrm{Adv}_{\mathtt{KE}}^{\mathtt{Game}_1}(\mathcal{A}) - \mathrm{Adv}_{\mathtt{KE}}^{\mathtt{Game}_0}(\mathcal{A})| \leq \mathrm{Adv}_{\mathtt{KE}}^{\mathtt{I2R}}(\mathcal{A})$$

The next game, $\mathtt{Game}_2$, attempts to guess the identities of the initiator and responder in the test session A^* and B^*, as well as which of A^*'s instances corresponds to the one used in the test session. To this end, it chooses two indexes $i_A, i_B \in \{1, \ldots, q\}$. It defines A^* and B^* to be the identities of the parties corresponding to these indices respectively (which will become known as during the game). It also chooses an index $j^* \in \{1, \ldots, q_s\}$, where q_s is an upper bound on the maximum number of session instances at a party. Suppose the test session query is on (A, i). The game declares the adversary a loser if the sid of the test session is not (A^*, B^*, c). Furthermore, if the initiator instance at A^* is not j^* the adversary also loses. It is easily to check that:

$$\mathrm{Adv}_{\mathtt{KE}}^{\mathtt{Game}_2}(\mathcal{A}) = 1/(q^2 q_s) \cdot \mathrm{Adv}_{\mathtt{KE}}^{\mathtt{Game}_1}(\mathcal{A}).$$

We are now in a position where we can use the adversary $\mathcal{A}$ in $\mathtt{Game}_2$ to construct an algorithm $\mathcal{B}_2$ which breaks the IND-CCA security of the encryption scheme. This algorithm operates as follows.

– It chooses two indexes $i_A, i_B \in \{1, \ldots, q\}$, where q is an upper bound on the number of parties participating. It defines A^* and B^* to be the identities of the parties corresponding to these indexes respectively.
– It chooses an index $j^* \in \{1, \ldots, q_s\}$ where q_s is an upper bound on the maximum number of session instances at any party.
– It sets $c^* := \perp$ (this is used for consistent decryption simulation).

- At the initialisation stage, it generates keys for all pairs according to the key distribution, except for the pair $\{A^*, B^*\}$.
- It simulates various oracles as follows.
- $\texttt{Send}(A, B, i, m)$: algorithm $\mathcal{B}_1$ proceeds as follows.
  - If the pair $\{A, B\}$ is different from $\{A^*, B^*\}$, then $\mathcal{B}_2$ knows $K_{A,B}$.
    - If $m = \lambda$, then A is the initiator, and $\mathcal{B}_2$ encrypts $(sk||t_A||B)$ for a random key sk using $K_{A,B}$. Here t_A is obtained by reading the current value of the $\texttt{Tick}$ oracle corresponding to A.
    - Otherwise, A acts as a responder. $\mathcal{B}_2$ processes the message using the key $K_{A,B}$, A's current $\texttt{Tick}$ value and the list L. It processes the list L according to protocol rules.
  - If the pair involved is $\{A^*, B^*\}$, algorithm $\mathcal{B}_2$ does not know K_{A^*,B^*}. It continues as follows.
    - Suppose $m = \lambda$. If $(A, B, i) = (A^*, B^*, j^*)$ algorithm $\mathcal{B}_2$ chooses two random keys sk_0^* and sk_1^* and calls the left-right encryption oracle on $(sk_0^*||t_A^*||B^*)$ and $(sk_1^*||t_A^*||B^*)$, where t_A^* is the current value of A's $\texttt{Tick}$ oracle. It then sets c^* to be the returned ciphertext, and outputs it as part of the message to be transmitted. Else it chooses a random key sk and calls the left-right encryption oracle with both entries set to $(sk||t_A||B)$, where t_A is the current time at A.
    - Suppose m contains $c^* \neq \perp$ and $(A, B) = (A^*, B^*)$. The ciphertext will be rejected as the identity under c^* will not match A, unless $A^* = B^*$. In this case it uses the next step.
    - Suppose m contains $c^* \neq \perp$ and $(A, B) = (B^*, A^*)$. In this case algorithm $\mathcal{B}_2$ simulates A's response by accepting it and terminating with $sid = (A^*, B^*, c^*)$ and $pid = A^*$, as long as the local time at B^* is within delta of t_A^* and a replay is not detected. It updates the list L at B^* according to the rules of the protocol.
    - Suppose m does not contain c^* (in particular this holds while c^* is $\perp$). Algorithm $\mathcal{B}_2$ uses the decryption oracle and processes the response using the local time and the list L. If accepted, it updates L. Note that all decryption queries made are legal.
- $\texttt{Corrupt}(A)$: algorithm $\mathcal{B}_2$ returns the keys it generated in the initialisation stage. If the query is on A^* or B^*, algorithm $\mathcal{B}_2$ returns a random bit and terminates. This termination is consistent with $\texttt{Game}_2$. If $\mathcal{B}_2$ should not do so, it must be the case that A^* and B^* are guessed correctly. However, in this case, this query is not allowed by the rules of the game, a contradiction.
- $\texttt{Reveal}(A, i)$: suppose this oracle has accepted (A, B, c). Algorithm $\mathcal{B}_2$ can decrypt c either using $K_{A,B}$, the decryption oracle or the identity under c^* unless $(A, B, c) = (A^*, B^*, c^*)$. In this case $\mathcal{B}_2$ terminates by returning a random bit. This response is consistent with $\texttt{Game}_2$ since, if $\mathcal{B}_2$'s answer was incorrect, it must be the case that *all* termination rules in this game have failed. An examination of this setting reveals that A^* and B^* are identities in the test session. The session identifier of the test session is (A^*, B^*, c^*) and no other oracles apart from A^* or B^* have accepted this. Furthermore this ciphertext is accepted at the test session as j^* was guessed correctly. But this query is forbidden by the rules of the original game.
- $\texttt{Tick}(A)$: these are answered by maintaining a counter for each party, as in the previous simulations.
- $\texttt{Test}(A, i)$: algorithm $\mathcal{B}_2$ terminates with a random bit if some termination rule holds. Otherwise, algorithm $\mathcal{B}_2$ returns the key sk_1^* generate above. Note that if all the termination rules fail, this answer is consistent with rules of $\texttt{Game}_2$.
- When $\mathcal{A}$ returns a bit b', algorithm $\mathcal{B}_2$ also returns this bit as its own guess.

Algorithm $\mathcal{B}_2$ simulates the environment in $\texttt{Game}_2$ perfectly for $\mathcal{A}$. By the rules of the game, the ciphertext accepted in the test session is that obtained from the left-right oracle in the IND-CCA game. Now, if the internal random bit of the IND-CCA game is 1, the key sk_1^* passed in the test session is the proper key, and if the bit is 0, it is a random key. Therefore the simulation has set (implicitly) the challenged bit of $\mathcal{A}$ to be its own challenge bit ($\mathcal{A}$ wins if it outputs 0 for a random key and 1 for the proper key). Therefore we have:

$$\mathrm{Adv}_{\texttt{KE}}^{\texttt{Game}_2}(\mathcal{A}) = \mathrm{Adv}_{\texttt{AuthEnc}}^{\texttt{IND-CCA}}(\mathcal{B}_2).$$

Putting the above together, and taking the inequalities in Appendix B into account, we obtain the desired result.

Finally, we now show that the protocol provides $(\beta + \delta)$-recent entity authentication from the initiator to the responder. We have already shown initiator-to-responder authentication. Hence, we have that when an honest responder oracle Π_B^j terminates, with overwhelming probability there exists a unique partner initiator oracle Π_A^i which has accepted at some point. The message that B received and validated contained a timestamp denoting $t_A(\texttt{acc}(A,i))$, i.e. the local time at A when A accepted. The β-synchronisation limitation on the adversary gives us that

$$|t_A(\texttt{acc}(A,i)) - t_B(\texttt{acc}(A,i))| \leq \beta.$$

Now, taking the local time $t_B(\texttt{term}(B,j))$ at B when termination occurred, and examining operation of the protocol, we have that the acceptance window mechanism guarantees that:

$$|t_B(\texttt{term}(B,j)) - t_A(\texttt{acc}(A,i))| \leq \delta.$$

Combining the two previous inequalities, we obtain:

$$|t_B(\texttt{term}(B,j)) - t_B(\texttt{acc}(A,i))| \leq \delta + \beta,$$

and the theorem follows. $\square$