Modelling Time, or A Step Towards Reduction-based Security Proofs for OTP and Kerberos

Jörg Schwenk
Horst Görtz Institute
Ruhr-University Bochum, Germany
joerg.schwenk@rub.de

ABSTRACT

The notion of time plays an important role in many practically deployed cryptographic protocols, ranging from One-Time-Password (OTP) tokens to the Kerberos protocol. However, time is difficult to model in a Turing machine environment.

We propose the first such model, where time is modelled as a global counter $\mathcal{T}$. We argue that this model closely matches several implementations of time in computer environments. The usefulness of the model is shown by giving complexity-theoretic security proofs for OTP protocols, HMQV-like one-round AKE protocols, and a variant of the basic Kerberos building block.

1. INTRODUCTION

Authentication.

Authentication is, besides key agreement, the most important security goal in cryptographic protocols. Loosely speaking, an authentication protocol is secure if the probability that an *active* adversary breaks the protocol is the same as for a *passive* adversary. For many cryptograhic protocols, authentication is of major importance: TLS is used to authenticate the server (e.g. through EV certificates), and One-Time-Password (OTP) schemes (e.g. SecureID) or Kerberos (and WWW variants like OpenID or SAML) have been designed as authentication protocols.

Authentication with nonces.

Replay attacks are one major threat for authentication protocols, so the *freshness* of a message must be guaranteed. In practical implementations, this is achieved by securely including "new" values into some protocol messages: either nonces chosen by the other party, or timestamps.

Nonces can easily be modelled in a Turing machine based environment: They are either read from a random input tape, or they are randomly generated by a probabilistic Turing machine. The security analysis then only has to take into account the probability distribution of these values. However, for one-sided authentication, this implies at least two messages, and for mutual authentication at least three messages, since each authenticating party has to receive back the nonce chosen by itself, and returned by the authenticated party.

Authentication with Timestamps.

Timestamps are more difficult to model, because time is not measurable for a Turing machine after it finished its computation, and waits for fresh input. (In practice, this is comparable to the problem of using time in smartcard computations: If the smart card is disconnected from power supply, it is no longer able to even increase a local clock.) However, timestamps enable us to design more efficient protocols, with less latency.

Reduction-based proofs in a Turing Machine model.

A Turing Machine (TM) is a mathematical model of a (von Neumann) computer. The famous Church-Turing thesis states that any computable algorithm can be implemented using a TM. Since a TM is a mathematical object, mathematical proof techniques can be applied, and these proofs form the basis of all theoretical computer science. On the other hand, subtle errors in the model may invalidate such proofs [20].

DEFINITION 1 (TURING MACHINE [20]). *A* Turing Machine $\mathcal{M}$ *is a tuple* $\mathcal{M} = (Q, \Gamma, b, \Sigma, \delta, q_0, F)$*, where* Q *is a finite, non-empty set of* states*,* Γ *is the* tape alphabet*,* $b \in \Gamma$ *is the* blank symbol*,* $\Sigma \subseteq \Gamma - \{b\}$ *is the input alphabet,* $q_0 \in Q$ *is the* initial state*,* $F \subseteq Q$ *is the set of* final *or* accepting states*, and* $\delta : (Q - F) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ *is a partial function called the* transition function*. A Turing machine is started by setting its internal state to* q_0*, writing the input string on the (infinite) tape, and positioning its read/write head on the first input symbol. As long as no accepting state is reached,* $\delta(s, q) = (s', q', m)$ *is evaluated, where* s *is the actual state of the TM, and* q *is the symbol read by the read/write head from the tape. As a result of this evaluation of* δ*, the internal state of* $\mathcal{M}$ *is set to* s'*, the symbol* q' *is written on the tape by overwriting* q*, anf the read/write head either moves right* ($m = R$) *or left* ($m = L$)*.*

Variants of Turing machines include probabilistic TMs (which in addition have a read-only random tape prefilled with a randomly chosen string) and interactive TMs (which in addition have a communication tape shared with another TM). All these variants are mathematically sound.

In complexity theory, large classes of efficiently computable (P) and efficiently decidable problems (NP) were defined by giving polynomial reductions to some well-known problems [18]. In cryptography, complex systems are shown to be secure by giving a polynomial reduction to a well-studied assumption (e.g. factoring of large integers, or $P \neq NP$). The basis for this research was the formalization of all computational processes as (probabilistic) Turing Machines (TM).

Research on reduction-based proofs for cryptographic protocols started with the seminal paper of Bellare and Rogaway [6] in 1993. Up to the best of our knowledge, randomness played an important role in all subsequent papers, but time stamps were never investigated.

Modelling Time.

Time plays an important role in IT security: X.509 certificates and other security tokens have a validity period, some One-Time-Password (OTP) systems need loosely synchronized clocks between hardware token and server, GPS Spoofing may only be detected by comparing internal clock values with the time contained in GPS signals, and malware may protect itself by querying external time sources.

However, running an internal (independent) clock always means measuring some physical parameter: movements of a pendulum, oscillation rate of a crystal, or an electronic transition frequency. *Thus the main problem with time is that we simply cannot model "real" time in a Turing Machine based model. All formal models claiming to be able to model time must therefore be carefully checked if they satisfy all restrictions imposed by Definition 1 or one of its variants. If they deviate from this strict formalism, the results may become invalid.*

Instead, we have to find a suitable approximation for time, which preserves the most important security guarantees offered by "real" time. [1]

If we look at one implementation of time, we get a motivation for the choice of our model: The Network Time Protocol (NTP, [32]) delivers the actual time to an application *on request*. Thus in our model we have chosen to implement time as a global counter $\mathcal{T}$, which is accessible to all Turing machines in our computing environment. If a fresh message has to be sent, the sending TM request a timestamp $ts = (t, aux)$ from $\mathcal{T}$. Upon reception of such a request, $\mathcal{T}$ first increases its local counter ($t \leftarrow t + 1$). Then the actual value t is returned, optionally with auxiliary data aux appended. This auxiliary data may e.g. be a digital signature, to prevent Denial-of-Service attacks by an adversary who would issue large values like $t + 2^k >> t$.

One can easily derive variants of this model, for example: Turing Machines may query $\mathcal{T}$ whenever they are activated, or even on each computation step they make. Each query may increase the counter, or a different pattern may be designed to increase it. The channel between $\mathcal{T}$ and the process oracles may be insecure, authentic, or even *untappable* [27], the latter guaranteing that the adversary may never influence the communication with the time source. From all these variants, we have chosen what we believe to be the simplest model, which however offers similar security guarantees than the other variants.

The Notion of Time-Security.

Our starting point is the definition of secure authentication protocols, given in the seminal paper by Bellare and Rogaway [6]. They motivated their definitions by introducing a *benign adversary*, who forwards all messages faithfully. They defined an authentication protocol to be secure if the winning probability of any adversary is (up to a negligible difference) equal to the winning probability of this benign adversary. They showed that this condition is, for many protocols using random nonces, equivalent to requiring that both parties only accept if they have *matching conversations* (cf. Definition 7). Our main goal is to find a replacement for the concept of *matching conversations*, since in one- and two-message protocols, this concept is not applicable: here the responder oracle *always* has a matching conversation to the initator oracle, but due to replay attacks active adversaries may influence the system significantly: With a benign adevrsary, there is at most one responder oracle that will accept on a single message; with an active adversary, there may be arbitrary many.

In this paper we only consider cryptographic protocols consisting of one or two messages. (If we have three messages, we can use random nonces, and achieve better security goals.) Thus there always is one oracle (responder) that has to decide whether to accept or reject after receiving a single message, and before (or without) sending a message. We will consider a protocol to be *time-secure*, if for each initiator oracle that has sent a message there is at most one responder oracle that accepts, and that this responder oracle will accept only if the message was forwarded unmodified by the adversary. The second goal can be achieved by using cryptographic primitives like message authentication codes or signature schemes, but for the first goal we need *timestamps*.

For two-message protocols, we can additionally base the acceptance condition for initiator oracles (which send and receive one message) on the classical notion of matching conversations (if a nonce is used, which is however *not* the case in all previously proposed two-message protocols), or we can also apply the notion of time-security here.

Scope of our results.

The results presented in this paper are directly applicable to OTP schemes based on counters, e.g. all OTP schemes based on the HOTP algorithm [34], which is the basis for many commercially deployed OTP schemes. HOTP uses an 8 Byte counter, a shared key between initiator and responder, and HMAC-SHA-1 as the MAC algorithm. The counter value at the initiator is incremented with every OTP generation, whereas the counter at the server is only incremented after a successful authentication, to the counter value used in the OTP. This exactly mirrors our time model. However, usability considerations have lead to the introduction of a truncation function, which reduces the entropy of the OTP significantly, and thus a throttling scheme must be used to prevent exhaustive MAC searches. These usability enhancements are out-of-scope here.

OTP schemes using loosely synchronized clocks (e.g. RSA

[1] Please note that some usecases of time may cause additional moidelling problems. E.g. the lifetime of a public key in an X.509 certificate is often limited, because with infinite lifetime, the adversary may compute the private key in non-polynomial time. To include this in a formal model, we would have to somehow couple the time value used in the protocol with the number of computation steps of the adversary.

SecureID or TOTP [35]) are based on clock counters, the main difference to HOTP being that the responder counter is increased independantly. We could modify the acceptance condition and the communication with $\mathcal{T}$ at the responder to include this in our model (see Appendix); however, in oder to get similar security properties, we would also have to add a second counter at the responder, in order to prevent replay attacks of OTPs within one given time step (default value is 30 seconds).

For two-message protocols, our results can be used to devise new protocols which may achieve explicit authentication for both initiator and responder, in addition to authenticated key exchange. With respect to the latter property, the model presented here is "weak" in the sense that it does not consider queries like *RevealDHExponent* or *RevealState* [16]; again, this is in order to keep the model (and the proofs) simple.

One important motivation for considering a hybrid definition of explicit authentication (using nonces and timestamps) was the fact that the two-party two-message building blocks of the Kerberos protocol (cf. Section 7) are of this type. In our model, we can show that these building blocks are authentication protocols, but we cannot cover the key exchange part of Kerberos: here a suitable theory for Single-Sign-On protocols is still missing.

Contribution.

The contributions of this paper are as follows:

- We propose the first theoretically consistent model for timestamps in cryptograhic protocols, which covers all Turing Machine-based implementations.
- We give a security definition for explicit authentication extending [6], applicable to a wide range of protocols.
- We show the usefulness of our definition by giving examples of secure one and two message protocols, together with formal security proofs.
- We sketch modified versions of the basic building block of the well-known Kerberos protocol (with time stamps, but without validity time ranges), for which security proofs (for authentication only) are possible in our model.

2. RELATED WORK

Timestamps.

An overview on usage and security problems of timestamps can be found in [29]. (See [19] for an updated version.) Motran et al. [33] use a different model for timestamping documents: they assume that a unique random string is broadcasted in each time period. An actual overview on hash-then-publish timestamping can be found in [11]. [26] shows that if malleable encryption (e.g. a streamcipher) is used, timestamp based authentication protocols may fail. [38] covers one-time passwords, but in the different context of password-based authenticated key exchange (PAKE).

Authenticated Key Exchange.

Reasearch on formal security models for two-party authentication and authenticated key exchange (AKE) protocols started with the seminal paper of Bellare and Rogaway [6]. In [7], 3-party session key distribution protocols were investigated, but authentication was omitted.

In the following years, research on cryptographic protocols focused either on authenticated group key exchange [28], or the higly efficient two-message protocols (see below). Explicit authentication was difficult to achieve in these protocols, thus variants of the AKE model introduced by Canetti and Krawczyk [13] were used.

Authenticated Key Exchange with Timestamps.

In a paper whose goals are closest to our work, Barbosa and Farshin [2] introduce two different models (one based on Canetti-Krawczyk and one following Bellare-Rogaway) to model AKE with timestamps, and a total of 6 theorems.

In [2], time is modeled as a local clock LocalTime, which is incremented by sending TICK requests. In comparison to our work (we use a global clock), [2] has a couple of drawbacks:

- There is no executional model which describes how local clocks are incremented and kept synchronized. Since desynchronization of local clocks is a major issue in practice, this is an important omission. (We use the global counter and received messages to synchronize.)
- In a secure authenticated key exchange protocol, the success probabilities of active and passive adversaries are roughly the same. For 3-message protocols using nonces, this was formalized using "matching conversations" in [6]. [2] tries to reformulate these definitions to the 1-message settings, but the important partnering definition is flawed: An active adversary may simply send a message generated by pi_A^i to pi_B^j and pi_B^{j+1}. Then according to [2], Definition 5, the three oracles are not partnered, thus the adversary may ask REVEAL to pi_B^j and at the same time TEST to pi_A^i, and thus always win the security game.
- The adversary is allowed to increment the local clock. This is counterintuitive: to name an extreme example, GPS spoofing can only be detected if a reliable local clock is available, despite all cryptographic protection mechanisms [41].
- Finally, the proof of Theorem 6, which looks similar to the results in our paper, doesn't make use of timestamps at all: The same security guarantees can be achieved if timestamps are replaced with sequence numbers, because sychronizity of local clocks is not modeled.

Two-message protocols.

In 1986, Matsumoto et al. [30] first studied implicitly authenticated Diffie-Hellman protocols, which results in a line of research generated many protocols. Research on this topic was re-initiated by the introduction of the MQV [31, 24], KEA1 [5] and HMQV [22] protocols. The latter is formally proven secure in a modified CK [13] model, where no explicit authentication is defined (only authenticated key exchange). A short overview on the employed models is given in [14].

Kerberos.

The Kerberos protocol [37] was introduced to the academic community in [39, 21]. First informal security investigations were published in [8, 17].

Formal analysis in Dolev-Yao-style models was started in [4, 3, 12]. Cremers and Mauw [15] analyzed protocols which were generalizations of the public-key and private-key Needham-Schroeder protocol [36], which was the model for Kerberos. Recently, Backes et al. [1] gave an analysis of the authentication and key secrecy properties of Keberos, and Li [25] gave an analysis of Kerberos in the Hoare logic.

Blanchetet al. [9] presents a proof on Kerberos that is somehow a mixture between Dolev-Yao and reduction-based proofs: they use assumptions from reduction based cryptography (e.g. IND-CCA2) to show that a set of ad-hoc security properties is fulfilled. However their proof is not a reduction in the sense that 'if one of these security properties is violated, then at least one of the assumptions can be broken".

Boldyreva and Kumar [10] provide valuable insights into the encryption schemes employed in Kerberos v5 by showing that under reasonable assumptions, a "modified general profile" can be shown to be IND-CCA and INT-TXT secure.

3. DEFINITIONS

Notation We use $\emptyset$ to denote the empty string, and $[n] = [1,n] = \{1,\ldots,n\} \subset \mathbb{N}$ for the set of integers between 1 and n. If A is a set, then we use $a \xleftarrow{\$} A$ to denote that a is drawn uniformly random from A. In case A is a probabilistic algorithm $a \xleftarrow{\$} A$ is used to denote that A returns a when executed with fresh random coins. We use κ to denote the security parameter. $x \xleftarrow{\$} \mathcal{A}(y)$ denotes the output x of the probabilistic algorithm $\mathcal{A}$ when run on input y and randomly chosen coins.

Digital Signature Schemes A digital signature scheme is a triple $\mathsf{SIG} = (\mathsf{SIG.Gen}, \mathsf{SIG.Sign}, \mathsf{SIG.Vfy})$, consisting of the key generation algorithm $(sk, pk) \xleftarrow{\$} \mathsf{SIG.Gen}(1^\kappa)$ generating a (public) verification key pk and a secret signing key sk on input of the security parameter κ, the signing algorithm $\sigma \xleftarrow{\$} \mathsf{SIG.Sign}(sk, m)$ generating a signature for message m, and the verification algorithm $\mathsf{SIG.Vfy}(pk, \sigma, m)$ returning 1, if σ is a valid signature for m under key pk, and 0 otherwise. Security is formalized in the following security game that is played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger generates an asymmetric key pair $(sk, pk) \xleftarrow{\$} \mathsf{SIG.Gen}(1^\kappa)$ and the public key pk is given to the adversary.
2. The adversary may adaptively query q messages m_i with $i \in [q]$ of his choice to the challenger. The challenger responds to each of these queries with a signature $\sigma_i = \mathsf{SIG.Sign}(sk, m_i)$ on m_i.
3. The adversary outputs a message/signature pair (m, σ).

Definition 2. *We say that* SIG *is* (t, ϵ, q)-secure *against* existential forgeries under adaptive chosen-message attacks *(EUF-CMA), if for all adversaries* $\mathcal{A}$ *that run in time* t *making at most* q *queries it holds that*

$$\Pr[(m,\sigma) \xleftarrow{\$} \mathcal{A}^{\mathcal{C}}(1^\kappa, pk) \text{ such that } \mathsf{SIG.Vfy}(pk, m, \sigma) = 1 \wedge m \notin \{m_1, \ldots, m_q\}] \leq \epsilon.$$

Symmetric Encryption Schemes A symmetric encryption (SE) scheme is a pair $\mathsf{SE} = (\mathsf{SE.Enc}, \mathsf{SE.Dec})$, consisting of the encryption algorithm $\mathsf{SE.Enc}(k, m)$ generating a ciphertext c for message m under key k, and the deterministic decryption algorithm $\mathsf{SE.Dec}(k, c)$ returning m, if c is a valid encryption and the error symbol $\perp$ otherwise. Security against chosen ciphertext attacks is formalized in the following security game that is played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The adversary may adaptively query the challenger for encryptions $\mathsf{SE.Enc}(k, m)$ of arbitrary plaintexts m, and for decryptions $\mathsf{SE.Dec}(k, c)$ of arbitrary ciphertexts c.
2. The adversary outputs two (fresh) messages m_0 and m_1 of the same length.
3. The challenger tosses coin $b \xleftarrow{\$} \{0,1\}$. It then sets $c = \mathsf{SE.Enc}(k, m_b)$ and sends c to the adversary.
4. The adversary may again adaptively query plaintexts m and ciphertexts c' of his choice, with the restriction that $m \notin \{m_0, m_1\}$ and $c' \neq c$.
5. Finally the adversary outputs a bit b'.

Definition 3. *We say that* SE *is* (t, ϵ, q)-secure *under* chosen ciphertext attacks *(CCA), if all adversaries* $\mathcal{A}$ *that run in time* t *making at most* q *encryption queries have advantage of at most* ϵ *to distinguish the ciphertext of* m_0 *from that of* m_1*, i.e.*

$$\left|\Pr\left[b = b'\right] - 1/2\right| \leq \epsilon.$$

Public Key Encryption Schemes A public key encryption (PKE) scheme is a triple $\mathsf{PKE} = (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$, consisting of the key generation algorithm $(sk, pk) \xleftarrow{\$} \mathsf{PKE.Gen}(1^\kappa)$ generating a (public) encryption key pk and a secret decryption key sk on input of the security parameter κ, the probabilistic encryption algorithm $\mathsf{PKE.Enc}(pk, m)$ generating a ciphertext c for message m, and the deterministic decryption algorithm $\mathsf{PKE.Dec}(sk, c)$ returning m, if c is a valid encryption and the error symbol $\perp$ otherwise. Security is formalized in the following security game that is played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger generates an asymmetric key pair $(sk, pk) \xleftarrow{\$} \mathsf{PKE.Gen}(1^\kappa)$ and the public key pk is given to the adversary.
2. The adversary may adaptively query the challenger for decryptions of arbitrary ciphertexts c. The challenger responds to each of these queries with the output of $\mathsf{PKE.Dec}(sk, c)$.
3. The adversary outputs a message m^*.
4. The challenger tosses coin $b \xleftarrow{\$} \{0,1\}$. It then sets $c_0 = \mathsf{PKE.Enc}(pk, m^*)$ and $c_1 = \mathsf{PKE.Enc}(pk, r)$ for a uniformly random message r that is of the same size as m^* and sends c_b to the adversary.
5. The adversary may again adaptively query ciphertexts c of his choice, now with the restriction that $c \neq c_b$. The challenger responds to each of these queries with the output of $\mathsf{PKE.Dec}(sk, c)$.

6. Finally the adversary outputs a bit b'.

DEFINITION 4. *We say that* PKE *is* (t, ϵ, q)-secure *under* adaptive chosen ciphertext attacks *(CCA), if all adversaries* $\mathcal{A}$ *that run in time* t *making at most* q *decryption queries have advantage of at most* ϵ *to distinguish the ciphertext of* m^* *from that of a truly random value, i.e.*

$$\left|\Pr\left[b = b'\right] - 1/2\right| \leq \epsilon.$$

Message Authentication Codes A *message authentication code* is an algorithm MAC. This algorithm implements a deterministic function $w = \mathsf{MAC}(K_{\mathsf{mac}}, m)$, taking as input a (symmetric) key $K_{\mathsf{mac}} \in \{0,1\}^{\kappa}$ and a message m, and returning a string w.

Consider the following security experiment played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger samples $K_{\mathsf{mac}} \xleftarrow{\$} \{0,1\}^{\kappa}$ uniformly random.

2. The adversary may query arbitrary messages m_i to the challenger. The challenger replies to each query with $w_i = \mathsf{MAC}(K_{\mathsf{mac}}, m_i)$. Here i is an index, ranging between $1 \leq i \leq q$ for some $q \in \mathbb{N}$. Queries can be made adaptively.

3. Eventually, the adversary outputs a pair (m, w).

DEFINITION 5. *We say that* MAC *is a* (t, ϵ, q)-secure *message authentication code, if for all adversaries* $\mathcal{A}$ *that run in time* t *and queries at most* q *messages holds that*

$$\Pr[(m,w) \xleftarrow{\$} \mathcal{A}^{\mathcal{C}}(1^{\kappa}) : w = \mathsf{MAC}(K_{\mathsf{mac}}, m) \wedge m \notin \{m_1, \ldots, m_q\}] \leq \epsilon_{\mathsf{MAC}}.$$

4. FORMAL MODEL

In this section, we describe the *execution environment*, where we try to model normal protocol execution, the *adversarial capabilities*, which describe against which type of adversary our protocol should be secure, and the *security model* which describes security games where the winning events correspond to a security breach of the protocol.

4.1 Execution Environment

Parties and process oracles.

We distinguish between *parties* $\{P_1, \ldots, P_n\}$ and process oracles π_i^s, $i \in \{1, \ldots, l\}$, run by party P_i. These process oracles and the party itself are modelled as Turing machines. Oracles can be initialized, can send and receive messages according to the protocol specification, and terminate either in `finished`, `accept` or `reject` state[2]. The complete set of states for oracles is $\Lambda = \{$`not_initialized`, `wait`, `finished`, `accept`, `reject`$\}$. In addition, we have two special parties $\mathcal{T}$ and $\mathcal{A}$.

Each party P_i has two local variables: A local time counter T_i, and a long-lived key k_i. This long-lived key is either a public key pair, or a list of $n-1$ symmetric keys shared with the other parties.

[2]We need an additional final `finished` state for one-message protocols because initiator oracles always have to reach a final state, but this can be no winning event for the adversary.

Long-lived keys of party P_i can be accessed for cryptographic operations, and the counter values t_i of P_i can be increased, by all process oracles (or oracles, for short) $\pi_i^1, ..., \pi_i^l$ of this party. Computed nonces, intermediate state values, and session keys are only known to a single oracle. If a session key is computed by an oracle π_i^s, the value k is stored in a variable K when the oracle enters `accept` or `finished` state. The current state of each oracle is stored in variable Λ, and the transcript of all messages sent and received (in chronological order) in variable $\mathsf{T}^{i,s}$.

There is one special party $\mathcal{T}$, which has a local counter T with actual state t; on request, this party increases the value of t by 1 and returns the actual value of t as a message $ts = (t, aux)$ over the network.

The adversary $\mathcal{A}$ is another special party which implements, as a Turing machine, a strategy to break the cryptographic protocol. The event(s) which define a protocol break are modelled as winning events in different games defined below.

Initiator and responder oracles.

Our security definition is focused on protocols with one and two messages; for protocols with three and more messages nonces can be used instead of timestamps, and the classical definitions and results from the Bellare-Rogaway and Canetti-Krawczyk models apply. For our security definitions, we need to distinguish between *initiator* and *responder* oracles.

DEFINITION 6. *An oracle who sends the first message in a protocol is called* initiator*, and an oracle who receives the first message is called* responder.

Increase local time on message reception.

Each party P_i has a local timer T_i with actual state t_i. Each time a message with a valid timestamp $ts = (t, aux)$ is received by some oracle π_i^s, the value t is compared to the actual state t_i. If $t \leq t_i$, the message is rejected. If $t > t_i$, the message is validated further, and t_i is replaced by the (greater) value t: $t_i \leftarrow t$. (This is how many smartcards handle time.) Only after this update of the local timer T_i, π_i^s may accept or send a message (if required by the protocol specification).

Retrieve actual time when sending a message.

When trying to send a message to P_j, π_i^s has no knowledge about the local time t_j. Therefore the oracle requests an (authenticated) new timestamp $ts = (t, aux)$ from $\mathcal{T}$, and compares it with T_i. If $t > T_i$, this value is securely included in the protocol message.

Remark: Without any interference from an active adversary, t is guaranteed to be strictly greater than all local time values. However, an active adversary controlling the network may replay old time values.

Non-authenticated Time.

In practice, timestamps may be signed, or they may be sent unauthenticated. For our model, we have chosen that timestamps are *not* authenticated. The reason for this is simple: The only attack that signed timestamps would prevent is a Denial-of-Service (DoS) attack, where the adversary would intercept the time request of an oracle, and return a

large time value $t^* = t + 2^k$. All parties receiving a valid message containing this time value will be blocked for 2^k time periods.

However, our model uses an active adversary with complete control on the network. Thus he can always perform DoS attacks, e.g. disconnect some party for 2^k time periods from the network. Thus the power of our adversary is not increased by omitting signatures.

In a weaker adversarial model however, e.g. where the adversary controls only well-defined parts of a communication network, authenticated timestamps may make sense.

Local states of oracles.

An initiator oracle π_i^s will, after being initiated, retrieve the actual time value t, prepare and send a message of the form[3] $(P_i, P_j, t, n_i^s, m, \sigma)$ where P_j denotes the identity of the intended receiving party, n_i^s a nonce chosen randomly by π_i^s, m the actual message, and σ the cryptographic protection of the message.

In a one-message protocol, π_i^s will immediately switch to `finished` state, and can no longer be activated; here the nonce may be omitted. A responder oracle π_j^t will be activated by a protocol message, which will be checked according to the protocol specification. If the check succeeds, π_j^t will switch to `accept` state.

In a two-message protocol, after sending the first message, an initiator oracle π_i^s will switch to a `wait` state to wait for a response message, and can also no longer be activated to send messages. A responder oracle π_j^t will be activated by a message, which will be checked according to the protocol specification. If the check succeeds, π_j^t will prepare a response message which includes the nonce from the received message, send this message, and switch to `accept` mode. Upon reception of a message from some responder, an initiator oracle will check if this message contains the nonce from the first message, and if the message is valid according to the protocol specification. If both criteria are fulfilled, it will switch to `accept` mode.

Matching Conversations.

Bellare and Rogaway [6] have introduced the notion of *matching conversations* in order to define correctness and security of an AKE protocol precisely. We adapt this notion and use it accordingly to define secure authentication for all following protocol types.

Recall, that $\mathsf{T}^{i,s}$ consists of all messages sent and received by π_i^s in chronological order (not including the initialization-symbol $\top$). We also say that $\mathsf{T}^{i,s}$ is the *transcript* of π_i^s. For two transcripts $\mathsf{T}^{i,s}$ and $\mathsf{T}^{j,t}$, we say that $\mathsf{T}^{i,s}$ is a *prefix* of $\mathsf{T}^{j,t}$, if $\mathsf{T}^{i,s}$ contains at least one message, and the messages in $\mathsf{T}^{i,s}$ are identical to and in the same order as the first $|\mathsf{T}^{i,s}|$ messages of $\mathsf{T}^{j,t}$.

Definition 7 (Matching conversations). *We say that π_i^s has a* matching conversation *to π_j^t, if*

- *$\mathsf{T}^{j,t}$ is a prefix of $\mathsf{T}^{i,s}$ and π_i^s has sent the last message(s), or*
- *$\mathsf{T}^{i,s} = \mathsf{T}^{j,t}$ and π_j^t has sent the last message(s).*

We say that two processes π_i^s and π_j^t have matching conversations *if π_i^s has a* matching conversation *to process π_j^t, and vice versa.*

4.2 Adversarial Capabilities

We assume that the adversary completely controls the network, including access to the time oracle $\mathcal{T}$. He may learn session keys, and he may learn some long-lived keys.

All this is modelled through queries. The Send query models that the adversary completely controls the network: All messages are sent to $\mathcal{A}$, who may then decide to drop the message, to store and replay it, or to alter and forward it. Thus messages received through a Send query are handled by the process oracles exactly like real protocol messages: They may be rejected, they may be answered, or they may start or terminate a protocol session. The Send message enables the adversary to initiate and run an arbitrary number of protocol instances, sequential or in parallel.

The Reveal and Corrupt queries model real world attacks: Brute force key searches or malware attacks on PC systems in case of the Reveal query, and e.g. sidechannel of fault attacks on smartcards in the case of the Corrupt query.

Finally the Test query is one important building block in defining security of a protocol (more precisely: key indistinguishability).

Formally stated:

- $\mathsf{Send}(\pi_i^s, m)$: The adversary can use this query to send any message m of his own choice to oracle π_i^s. The oracle will respond according to the protocol specification, depending on its internal state. If $m = \top$ consists of a special symbol $\top$, then π_i^s will respond with the first protocol message.
- $\mathsf{Reveal}(\pi_i^s)$: Oracle π_i^s responds to a Reveal-query with the contents of variable K. Note that we have $k \neq \emptyset$ only if $\Lambda \in \{\texttt{accept}, \texttt{finished}\}$.
- $\mathsf{Corrupt}(P_i)$: This query is used in a public key setting. Oracle π_i^1 responds with the private key sk_i from the long-term key pair $k_i = (pk_i, sk_i)$ of party P_i. Then the party and all its oracles are marked as corrupted.
- $\mathsf{Corrupt}(P_i, j)$: This query is used in symmetric key settings. Oracle π_i^1 responds with the long-term secret key $k_{i,j}$ shared between party P_i and party P_j. Then this key is marked as corrupted.
- $\mathsf{Test}(\pi_i^s)$: This query may only be asked once throughout the game. Oracle π_i^s handles this query as follows: If the oracle has state $K = \emptyset$, then it returns some failure symbol $\perp$. Otherwise it flips a fair coin b, samples a random elem'ent $k_0 \xleftarrow{\$} \mathcal{K}$, sets $k_1 = k$ to the 'real' session key, and returns k_b.

Definition 8. *An oracle is called* corrupted *if it is marked as corrupted (public key case), or if it uses a corrupted long-term key in its computations (symmetric case).*

4.3 Security Model 1: One-message Protocols

One-message protocols like OTP are used for explicitly authenticating a party. Thus we have to find a definition for secure authentication taking into account the ideas from [6], and the pecularities of our definition of time.

[3]The exact structure of the message is specified for each protocol separately.

A *benign* adversary $\mathcal{A}$ is an adversary that forwards messages without modifying them. We would like to define a secure authentication protocol as a protocol where the winning probability of any adversary equals, up to a negligible difference, the winning probability of a benign adversary.

Freshness vs. validity time periods.

One major difference to [6], due to our definition of time, is the fact that even with a benign adversary there are situations where the responder oracle may not accept: Assume there are two different initiator oracles $\pi_{A_1}^{s_1}$ and $\pi_{A_2}^{s_2}$, who both intend to send a message to party B. They retrieve two time values t_1, t_2 from T, with $t_2 = t_1 + 1$. Now if $\pi_{A_2}^{s_2}$ is the first oracle who actually sends a message to B, then some responder oracle π_B^t will accept this message, and increase $T_B := t_2$. If now $\pi_{A_1}^{s_1}$ sends its message, it will be rejected to to the fact that $t_1 < T_B$. Please note that this may happen even if a benign adersary also forwards all messages in the same order as received.

Real-world protocols avoid this problem by defining validity time periods: A message will be accepted if it is not older than, say, 5 minutes, or if it was received at a time that is within the timeframe explicitely mentioned in the message. However, including validity time periods in the model poses different problems: either replay attacks will become possible during the validity time frame and we must change our security definitions, or we have to introduce an additional counter at each party for each other party to exclude these replay attacks (see Appendix).

Thus, for the sake of clarity, we decided to keep the model of time simple, and to accept the consequence that even with benign adversaries, responder oracles may not accept.

Replay attacks.

The main class of attack we want to protect against is *replay attacks*. In a replay attack, $\mathcal{A}$ may intercept a message sent by a sending oracle, and forward it to two or more receiving oracles. We would like to call a protocol secure if at most one of these receiving oracles accepts, because this is exactly what we can expect from a benign adversary. To achive this goal across several parties, we have to include the identity of the receiving party in the first message.

Security Game G_{A-1}**.** In this game, the challenger $\mathcal{C}$ sets up a protocol environment with n parties $P_1, ..., P_n$, and prepares l protocol oracles $\pi_i^1, ..., \pi_i^l$ for each party. If initiated by the adversary by a special start message, these oracles act as initiator oracles, and if initiated with a normal protocol message, they act as responder oracles. $\mathcal{C}$ generates long-lived keys (or long-lived key pairs, resp.) for each party (for each pair of parties, resp.), and simulates the time oracle $\mathcal{T}$. T and all counters $T_i, i \in \{1, ..., n\}$ are initialized to 0.
$\mathcal{A}$ may now ask up to q Send and Corrupt queries. $\mathcal{A}$ wins the game if there are at least two responder oracles that accept the same message, or if there is a responder oracle that accepts a message from an uncorrupted expected sender which has not been issued by any sender oracle.

DEFINITION 9. *A protocol* Π *is a* (τ, q, ϵ) time-secure *authentication protocol, if for each adversary* $\mathcal{A}$ *that runs in time* τ *and asks at most* q *queries, with probability at least* $1 - \epsilon$ *we have that*

1. *for each responder oracle that accepts, there is exactly one uncorrupted finished initiator oracle, and*
2. *for each finished initiator oracle, there is at most one responder oracle that accepts.*

Authenticated Key Exchange.

We will now extend the definition to key exchange. Here each session key should be indistinguishable from a randomly chosen key, for any adversary.

Security Game G_{AKE-1}**.** The setup of this game is the same as in G_{A-1}. In addition, the adversary is allowed to ask upt to $q - 1$ Reveal queries, and one Test query to an oracle π_i^s, subject to the following conditions:

- If π_i^s is an initiator, then no Reveal query may be asked to any responder oracle receiving the message from π_i^s.
- If π_i^s is an responder, then no Reveal query may be asked to the unique initiator oracle that has sent the message, and the party of this initiator oracle ust be uncorrupted.

At the end of the game, $\mathcal{A}$ issues a bit b' and wins the game if $b = b'$, where b is the bit chosen by π_i^s in answering the Test query.

DEFINITION 10. *A one-message protocol* Π *is a* (τ, q, ϵ) time-secure *authenticated key exchange protocol, if* Π *is a* (τ, q, ϵ) *time-secure authentication protocol in the sense of Definition 9, and if for each adversary* $\mathcal{A}$ *that runs in time* τ *and asks at most* q *queries, we have that that*

$$Adv_{\mathcal{A}} = |Pr(b = b') - 1/2| \leq \epsilon$$

in Game G_{AKE-1}.

4.4 Security Model 2: Two-message Protocols

With the publication of HMQV [23] research on one-round key agreement protocols intensified. However, since the responder is always subject to replay attacks, the notion of *authentication protocol* developed in [6] was given up, and was replaced by a new definition for *authenticated key exchange* proposed by Canetti and Krawczyk [13].

In this section, we try to give a new definition of a secure authentication protocol, by combining timestamps and nonces. We use this hybrid approach since this is a building block of the Kerberos protocol, where several two-message protocols of this kind are combined.[4]

Remark: Please note that this definition does not apply to two-message protocols currently discussed in the literature, because most of them neither use timestamps nor nonces.

Authentication.

[4]It should be clear that we also can achive mutual authentication by simply applying the one-message protocol of Section 5 in both directions, using the model for authentication proposed there.

Security Game G_{A-2}**.** In this game, the setup is identical to Game G_{A-1}.
$\mathcal{A}$ may now ask up to q Send and Corrupt queries. $\mathcal{A}$ wins the game if there are at least two responder oracles that accept the same message, if there is a responder oracle that accepts a message from an uncorrupted expected sender which has not been issued by any sender oracle, or if there is a initiator oracle that accepts without having a matching conversation with a responder.

DEFINITION 11. *A two-message protocol* Π *is a* (τ, q, ϵ) time-nonce-secure *authentication protocol, if for each adversary* $\mathcal{A}$ *that runs in time* τ *and asks at most* q *queries, with probability at least* $1 - \epsilon$ *we have that*

1. *for each responder oracle that accepts, there is exactly one uncorrupted finished initiator oracle,*
2. *for each finished initiator oracle, there is at most one responder oracle that accepts, and*
3. *for each initiator oracle that accepts, there is exactly one responder oracle that has a matching conversation.*

Authenticated Key Exchange.

We will again extend the definition to key exchange. Here each session key should be indistinguishable from a randomly chosen key, for any adversary.

Security Game G_{AKE-2}**.** The setup of this game is the same as in G_{A-1}. In addition, the adversary is allowed to ask up to $q - 1$ Reveal queries, and one Test query to an uncorrupted oracle π_i^s, subject to the following conditions:

- If π_i^s is an initiator, then no Reveal query may be asked to any responder oracle receiving the message from π_i^s, and all responder oracles must be uncorrupted.
- If π_i^s is an responder, then no Reveal query may be asked to the unique initiator oracle that has sent the message, and the party of this initiator oracle must be uncorrupted.

At the end of the game, $\mathcal{A}$ issues a bit b' and wins the game if $b = b'$, where b is the bit chosen by π_i^s in answering the Test query.

DEFINITION 12. *A two-message protocol* Π *is a* (τ, q, ϵ) time-nonce-secure *authenticated key exchange protocol, if* Π *is a* (τ, q, ϵ) *time-nonce-secure authentication protocol in the sense of Definition 11, and if for each adversary* $\mathcal{A}$ *that runs in time* τ *and asks at most* q *queries, we have that*

$$Adv_{\mathcal{A}} = |Pr(b = b') - 1/2| \leq \epsilon$$

in Game G_{AKE-2}.

5. ONE-TIME-PASSWORDS

In a One-Time-Password (OTP) protocol, an initiator (party A) wants to authenticate against a responder (party B). To achieve this goal, A requests a timestamp from $\mathcal{T}$, and appends a cryptographic checksum to the message consisting of the identifiers of A and B, and the timestamp t. Typically, symmetric cryptography is used in OTP protocols. Thus in our example, we assume that a message authentication code MAC is used for this purpose. The OTP protocol is defined in Figure 5.

THEOREM 1. *If* MAC *is a* $(\tau_{MAC}, \epsilon_{MAC}, q_{MAC})$ *secure message authentication code, then* Π_{OTP} *is a* (τ, q, ϵ) *time-secure authentication protocol with* $\tau_{MAC} \approx \tau$, $q_{MAC} \approx q$, *and*

$$\epsilon \leq n^2 \cdot \epsilon_{MAC}.$$

PROOF. The proof is modelled as a short sequence of games. Game G_0 is the original game, where our adversary tries to force an oracle to accept with $otp*$ which is either faked, or has already been accepted by a different oracle. Thus we have

$$\epsilon_0 = \epsilon.$$

In G_1, we guess the initiator party A, and the receiver party B. They share a common symmetric key k_{AB}. Our guess is that the adversary will succeed in making one receiver oracle π_B^t accept either a fake OTP $otp*$, or that two oracles π_B^t and $\pi_B^{t'}$ will accept the same OTP otp. If our guess is wrong, we abort the game, and the adversary looses. Thus his winning probability is reduced by a factor n^2.

$$n^2 \cdot \epsilon_1 = \epsilon_0.$$

In G_2, we abort the game if the adversary $\mathcal{A}$ forges a valid message authentication code MAC for key k_{AB}. This may happen only with probability ϵ_{MAC}, since we could use such an adversary to break the security of the MAC scheme: The simulator replaces all MAC computations involving the key k_{AB} with calls to a MAC challenger $\mathcal{C}_{\mathsf{MAC}}$ which uses a randomly chosen MAC key k; if $\mathcal{A}$ forges a valid message authentication code mac for a fresh otp message which has not been queried from the MAC challenger, then we have broken the MAC challenge. Thus we have

$$\epsilon_1 \leq \epsilon_2 + \epsilon_{\mathsf{MAC}}.$$

Since we have excluded MAC forgeries in this game, we are left with OTPs $otp = (C, S, ts, \mathsf{MAC}(k_{CS}, C.S.ts))$ which were generated by non-corrupted oracles, where only the value ts may be influenced by the adversary. Thus condition 1 of Definition 9 is always true (i.e. holds with probability 1), and we are left with condition 2.

If $\mathcal{A}$ tries to send otp to any oracle of a party $T \neq S$, T will not accept because the target identity is different. If A tries to send otp to oracle π_C^s, but otp has already been accepted by π_C^t, π_C^s will not accept because $ts \leq t_C$. Thus also condition 2 is always fulfilled (i.e. holds with probability 1) we have

$$\epsilon_2 = 0.$$

□

6. ONE-ROUND AUTHENTICATION PROTOCOLS

We first give an example of a two-message mutual authentication protocol without key exchange, to exemplify the use of our model. In the second part of this section, we will extend this to authenticated key exchange, where the session key is chosen by the responder. This special choice is made to closely match the building blocks of Kerberos, and can easily be extended to other key exchange methods.

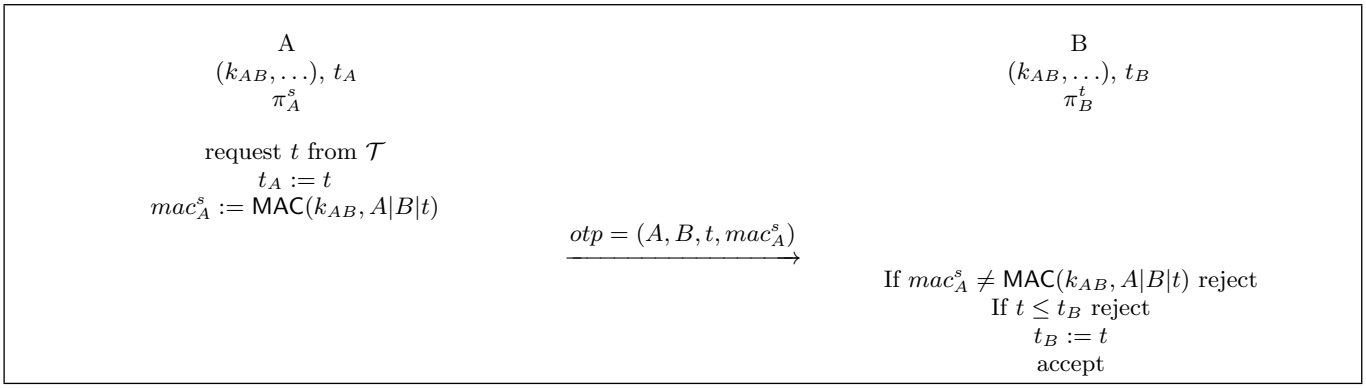

Figure 1: The One-message Authentication protocol Π_{OTP}.

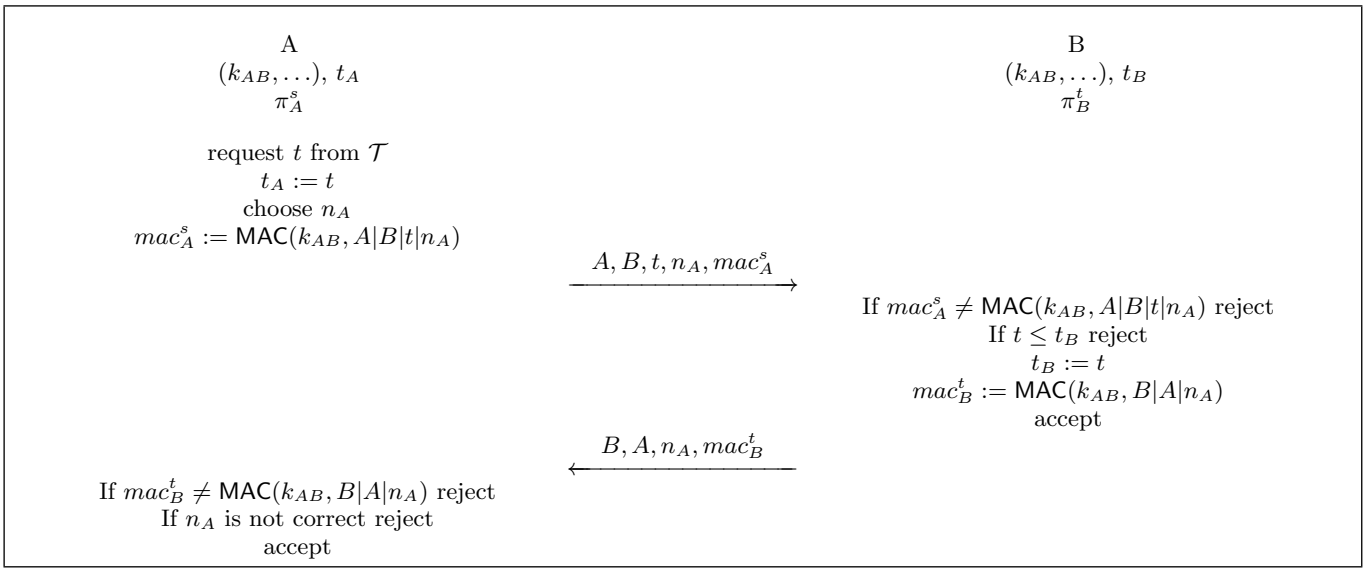

Figure 2: The Two-message Authentication protocol Π_{H2A}.

6.1 Authentication

THEOREM 2. *If* MAC *is a* $(\tau_{MAC}, \epsilon_{MAC}, q_{MAC})$ *secure message authentication code, then the* Π_{H2A} *protocol defined in Figure 6.1 is a* (τ, q, ϵ) *time-nonce-secure authentication protocol with respect to Definition 11 with* $\tau \approx \tau_{MAC}$, $q \approx q_{MAC}$, *and*

$$\epsilon \leq n^2 \cdot \epsilon_{MAC} + n^2 l \cdot (\frac{(nl)^2}{2^\lambda} + \epsilon_{MAC}).$$

PROOF. Adversary $\mathcal{A}$ can win the game by either making an initiator oracle accept, or a responder oracle, or both. Thus we have

$$\epsilon \leq \epsilon_I + \epsilon_R.$$

Since we can apply the proof of Theorem 1 to the responder oracle, we have $\epsilon_R \leq n^2 \cdot \epsilon_{MAC}$.

Thus we are left with the proof for the initiator oracle. The proof is modelled as a short sequence of games. Game G_0 is the original game, but the adversary only wins this game if an initiator oracle to accepts. Thus we have

$$\epsilon_0 = \epsilon_I.$$

In G_1, we guess the initiator oracle π_A^s that will accept, and the responder party B. They share a common symmetric key k_{AB}. If our guess is wrong, we abort the game, and the adversary looses. Thus his winning probability is reduced by a factor $l \cdot n^2$.

$$l \cdot n^2 \cdot \epsilon_1 = \epsilon_0.$$

In G_2, the simulator replaces all computations with the key k_{AB} with a MAC oracle. We abort the game if the adversary $\mathcal{A}$ forges a valid message authentication code MAC for this oracle. Thus we have

$$\epsilon_1 \leq \epsilon_2 + \epsilon_{\mathsf{MAC}}.$$

In G_3, we abort the game if two oracles choose the same nonce. The probability for this is bounded above by $\frac{(nl)^2}{2^\lambda}$. Thus we have

$$\epsilon_2 \leq \epsilon_3 + \frac{(nl)^2}{2^\lambda}.$$

Since we have excluded MAC forgeries and nonce collisions, $\mathcal{A}$ can only make an initiator oracle accept if he forwards a message from one of the (possibly many) responder oracles that have received the message from the initator oracle containing the chosen nonce. Thus there is a matching conversation, and we have

$$\epsilon_3 = 0.$$

□

6.2 Authenticated Key Exchange

The novel authenticated key exchange protocol presented in Figure 6.2 is modelled after the building blocks of the Kerberos protocol: authentication and session key encryption are based on symmetric cryptography, and the session key is chosen by the responder (the Kerberos server). Variants of this protocol are used three times sequentially in Kerberos, with the same initiator (the Kerberos client), but different responders (Ticket Granting Ticket Server, Ticket Server, Server).

THEOREM 3. *If* MAC*is a* $(\tau, \epsilon_{MAC}, q_{MAC})$ *secure message authentication code, and* SE *is a* $(\tau, \epsilon_{SE}, q_{SE})$ *secure symmetric encryption scheme, then the* Π_{ORAKEl} *protocol defined in Figure 6.2 is a* (τ, q, ϵ) *time-secure authented key exchange protocol with respect to Definition 12 with*

$$\epsilon \leq n^2 \cdot \epsilon_{MAC} + ln^2(\frac{(nl)^2}{2^\lambda} + \epsilon_{MAC}) + n^2 \cdot \epsilon_{SE}.$$

PROOF. Adversary $\mathcal{A}$ can win the game by either breaking the acceptance condition, or key indistinguishability, or both. Thus we have

$$\epsilon \leq \epsilon_A + \epsilon_{KE}.$$

Since we can apply the proof of Theorem 2 to the responder oracle, we have $\epsilon_A \leq n^2 \cdot \epsilon_{MAC} + ln^2(\frac{(nl)^2}{2^\lambda} + \epsilon_{MAC})$.

Thus we are left with the proof for key indisinguishability. In Game G_1, we first have to guess the symmetric key k_{AB}^e that will be used to encrypt the session key for the test oracle. If we guessed wrong, the adversary loses the game. Thus we loose a factor n^2 in this game.

In Game G_2, we replace all computations involving the key k_{AB}^e with our SE-challenger. Since $\mathcal{A}$ is not allowed to corrupt the key used in the test session, we can still simulate all protocol messages. Now if $\mathcal{A}$ is able to distinguish real from random keys, our SE-challenger is able to break the CCA-security of the symmetric encryption scheme. □

7. KERBEROS

It remains unclear why the three message pairs in Kerberso (see Appendix B) should have a different structure. Thus we will give a modified two-party building block in Figure 7 which can be substituted for any two-party block in the original Kerberos protocol, without changing the protocol flow or the initial key setup.

We are able to prove that this modified building-block consists of a one-message time-secure key exchange protocol, and a two-message time-nonce-secure authentication protocol. By concatenating these building blocks, the Kerberos protocol can be shown to be secure.

THEOREM 4. *The one-message protocol consisting of message* m_K *only is a time-secure authenticated key exchange protocol between* K *and* S *in the sense of Definition 10.*

PROOF. (Sketch) The MAC construction guarantees that for each accepting oracle, there must be exactly one uncorrupted oracle that has sent the message. The timestamp guarantees that there is at most one accepting S-oracle, and the encryption prevents the adversary from getting any information on the ciphertext. □

THEOREM 5. *If executed after the reception of message* m_K *by* K, *the two-message protocol consisting of messages* m_C *and* m_S *is a time-nonce-secure authentication protocol between* C *and* S *in the sense of Definition 11.*

PROOF. (Sketch) The proof is similar to the proof of Theorem 2. □

8. CONCLUSION AND FUTURE WORK

In this paper we have presented a first simple formal model to prove the security of timestamp-based authentication protocols, with reduction-based proofs, in a Turing Machine

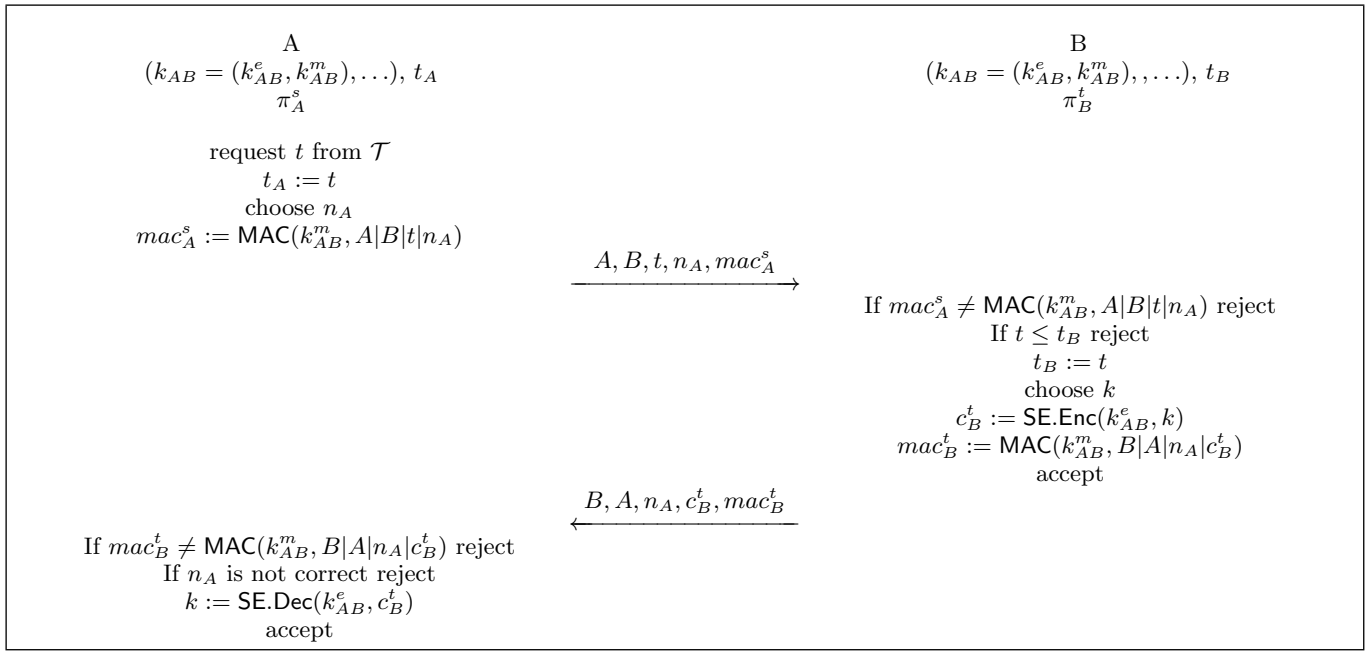

Figure 3: The One-Round Authenticated Key Exchange ($\mathcal{ORAKE}1$) protocol.

K	$k = (k^e, k^m)$, $k_C = (k^e_C, k^m_C)$, t_K
C	$k_C = (k^e_C, k^m_C)$, t_C
S	$k = (k^e, k^m)$, t_S
$\pi^{s_K}_K$	request t from $\mathcal{T}$; $t_K := t$
	$c_K := \mathsf{SE.Enc}(k^e, k_C)$
	$mac_K := \mathsf{MAC}(k^m, K\|\|S\|\|t_K\|\|C\|\|c_K)$
	$m_K := (K, S, t_K, C, c_K, mac_K)$
$\pi^{s_K}_K \rightarrow \pi^{s_C}_C$	m_K
$\pi^{s_C}_C$	request t from $\mathcal{T}$; $t_C := t$
	choose n_C
	$mac_C := \mathsf{MAC}(k^m_C, C\|\|S\|\|t_C\|\|n_C)$
	$m_C := (C, S, t_C, n_C, mac_C)$
$\pi^{s_C}_C \rightarrow \pi^{s_S}_S$	m_K, m_C
$\pi^{s_S}_S$	check mac_K on m_k
	if $t_K \leq t_S$ reject; $t_S := t_K$
	$k_C := \mathsf{SE.Dec}(k^e, c_K)$
	check mac_C on m_C with k^m_C
	if $t_C \leq t_S$ reject; $t_S := t_C$
	accept
	$mac_S := \mathsf{MAC}(k^m_C, S\|\|C\|\|n_C)$
	$m_S := (S, C, n_C, mac_S)$
$\pi^{s_S}_S \leftarrow \pi^{s_C}_C$	m_S
$\pi^{s_S}_S$	check mac_S on m_S with k^m_C
	if n_C incorrect reject
	accept

Figure 4: A Generic Kerberos bulding block.

environment. We tried to formalize the security goal that replay attacks are prevented by timestamps.

This model can be exended in various directions: (1) Most time-based security infrastructures or protocols use validity time frames, which can be modelled by allowing the responder to query the time oracle $\mathcal{T}$ on reception of a message. However, precautions must be taken to disallow replay attacks. (2) If time is involved, the power of an active adversary should be restricted to get more realistic security models: currently, DoS attacks by the adversary cannot be prevented.

The analysis of a building block of Kerberos given here is far from being a complete reduction based proof for Kerberos. Here a model is needed that formalizes the role of the trusted third Kerberos party in each basic three-party step, a model that may be useful in analyzing the varient of SSO protocols being proposed today.

9. REFERENCES

[1] Michael Backes, Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, and Joe-Kai Tsay. Cryptographically sound security proofs for basic and public-key kerberos. pages 362–383, 2006.

[2] Manuel Barbosa and Pooya Farshim. Security analysis of standard authentication and key agreement protocols utilising timestamps. In Bart Preneel, editor, *AFRICACRYPT*, volume 5580 of *Lecture Notes in Computer Science*, pages 235–253. Springer, 2009.

[3] Giampaolo Bella and Lawrence C. Paulson. Kerberos version 4: Inductive analysis of the secrecy goals. In Jean-Jacques Quisquater, Yves Deswarte, Catherine Meadows, and Dieter Gollmann, editors, *ESORICS*, volume 1485 of *Lecture Notes in Computer Science*, pages 361–375. Springer, 1998.

[4] Giampaolo Bella and Elvinia Riccobene. Formal

analysis of the kerberos authentication system. *J. UCS*, 3(12):1337–1381, 1997.

[5] Mihir Bellare and Adriana Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. pages 273–289, 2004.

[6] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. pages 232–249, 1994.

[7] Mihir Bellare and Phillip Rogaway. Provably secure session key distribution: The three party case. pages 57–66, 1995.

[8] Steven M. Bellovin and Michael Merritt. Limitations of the kerberos authentication system. In *USENIX Winter*, pages 253–268, 1991.

[9] Buno Blanchet, Aaron D. Jaggard, Andre Scedrov, and Joe-Kai Tsay. Computationally sound mechanized proofs for basic and public-key Kerberos. pages 87–99, 2008.

[10] Alexandra Boldyreva and Virendra Kumar. Extended abstract: Provable-security analysis of authenticated encryption in kerberos. pages 92–100, 2007.

[11] Ahto Buldas and Margus Niitsoo. Optimally tight security proofs for hash-then-publish time-stamping. In Steinfeld and Hawkes [40], pages 318–335.

[12] Frederick Butler, Iliano Cervesato, Aaron D. Jaggard, and Andre Scedrov. A formal analysis of some properties of kerberos 5 using msr. In *CSFW*, pages 175–. IEEE Computer Society, 2002.

[13] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. pages 453–474, 2001.

[14] Kim-Kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock. Examining indistinguishability-based proof models for key establishment protocols. In Bimal K. Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 585–604. Springer, 2005.

[15] Cas Cremers and Sjouke Mauw. A family of multi-party authentication protocols - extended abstract. In *Proc. 1st Benelux Workshop on Information and System Security (WISSEC) 2006*, 2006. `http://www.cosic.esat.kuleuven.be/wissec2006/papers/9.pdf`.

[16] Cas J. F. Cremers. Session-state reveal is stronger than ephemeral key reveal: Attacking the naxos authenticated key exchange protocol. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS*, volume 5536 of *Lecture Notes in Computer Science*, pages 20–33, 2009.

[17] Bryn Dole, Steven W. Lodin, and Eugene H. Spafford. Misplaced trust: Kerberos 4 session keys. In *NDSS*. IEEE Computer Society, 1997.

[18] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[19] Stuart Haber and Henri Massias. Time-stamping. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security (2nd Ed.)*, pages 1299–1303. Springer, 2011.

[20] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 3rd edition, 2006.

[21] John T. Kohl. The use of encryption in kerberos for network authentication. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 35–43. Springer, 1989.

[22] Hugo Krawczyk. Hmqv: A high-performance secure diffie-hellman protocol. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566. Springer, 2005.

[23] Hugo Krawczyk. HMQV: A high-performance secure diffie-hellman protocol. pages 546–566, 2005.

[24] Laurie Law, Alfred Menezes, Minghua Qu, Jerome A. Solinas, and Scott A. Vanstone. An efficient protocol for authenticated key agreement. *Des. Codes Cryptography*, 28(2):119–134, 2003.

[25] Qin Li, Fan Yang, Huibiao Zhu, and Longfei Zhu. Formal modeling and analyzing kerberos protocol. In *Proc. 2009 WRI World Congress on Computer Science and Information Engineering*, volume 7, pages 813–819. IEEE, 2009.

[26] Zhan Liu and Mi Lu. Authentication protocols with time stamps: – encryption algorithm dependent. In Hamid R. Arabnia, editor, *International Conference on Internet Computing*, pages 81–86. CSREA Press, 2006.

[27] Emmanouil Magkos, Mike Burmester, and Vassilios Chrissikopoulos. Receipt-freeness in large-scale elections without untappable channels. In Beat Schmid, Katarina Stanoevska-Slabeva, and Volker Tschammer, editors, *I3E*, volume 202 of *IFIP Conference Proceedings*, pages 683–694. Kluwer, 2001.

[28] Mark Manulis. Provably secure group key exchange, 2007.

[29] Henri Massias, X. Serret Avila, and Jean-Jacques Quisquater. Timestamps: Main issues on their use and implementation. In *WETICE*, pages 178–183. IEEE Computer Society, 1999.

[30] Tsutomu MATSUMOTO, Youichi TAKASHIMA, and Hideki IMAI. On seeking smart public-key-distribution systems. *IEICE TRANSACTIONS*, E69-E No.2:pp.99–106, 1986/02/20.

[31] Alfred Menezes, Minghua Qu, and Scott A. Vanstone. Some new key agreement protocols providing mutual implicit authentication. *SecondWorkshop on Selected Areas in Cryptography (SAC 95)*, 1995.

[32] D. Mills, J. Martin, J. Burbank, and W. Kasch. Network Time Protocol Version 4: Protocol and Algorithms Specification. RFC 5905 (Proposed Standard), June 2010.

[33] Tal Moran, Ronen Shaltiel, and Amnon Ta-Shma. Non-interactive timestamping in the bounded-storage model. *J. Cryptology*, 22(2):189–226, 2009.

[34] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen. HOTP: An HMAC-Based One-Time Password Algorithm. RFC 4226 (Informational), December 2005.

[35] D. M'Raihi, S. Machani, M. Pei, and J. Rydell. TOTP: Time-Based One-Time Password Algorithm. RFC 6238 (Informational), May 2011.

[36] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of

computers. *Communications of the ACM*, 21(21):993–999, December 1978.

[37] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (V5). RFC 4120 (Proposed Standard), July 2005. Updated by RFCs 4537, 5021, 5896, 6111, 6112, 6113, 6649, 6806.

[38] Kenneth G. Paterson and Douglas Stebila. One-time-password-authenticated key exchange. In Steinfeld and Hawkes [40], pages 264–281.

[39] Jennifer G. Steiner, B. Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. In *USENIX Winter*, pages 191–202, 1988.

[40] Ron Steinfeld and Philip Hawkes, editors. *Information Security and Privacy - 15th Australasian Conference, ACISP 2010, Sydney, Australia, July 5-7, 2010. Proceedings*, volume 6168 of *Lecture Notes in Computer Science*. Springer, 2010.

[41] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, and Srdjan Capkun. On the requirements for successful gps spoofing attacks. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 75–86. ACM, 2011.

APPENDIX

A. MODELING VALIDITY TIME INTERVALS

To be able to model validity time intervals, the model can be modified as follows.

Retrieve actual time when sending a message.

When trying to send a message to P_j, π_i^s requests a new timestamp $ts = (t, aux)$ from $\mathcal{T}$, and compares it with T_i. If $t > t_i$, the oracle calculates another value $t_{exp} := t + n_{exp}$. Both t and t_{exp} are securely included in the protocol message.

Increase local time on message reception.

Each party P_i has a local timer T_i with actual state t_i. Each time a message with a timestamp t_j is received from some oracle π_j^t, the oracle π_i^s receiving this message compares the value t_j to the actual state t_i of T_i. If $t_j \leq t_i$, the message is rejected. If $t_j > t_i$, t_i is replaced by the (greater) value t_j: $t_i \leftarrow t_j$, and the message is validated further. Now π_i^s requests a new timestamp $ts = (t, aux)$ from $\mathcal{T}$, and compares it with t_{exp} from the message. If $t_{exp} \leq t$, the message is rejected, else $t_i \leftarrow t$ and the message is validated further.

B. KERBEROS OVERVIEW

The Kerberos protocol is a four-party protocol, executed between a client C, a Key Distribution Center KDC, a Ticket Granting Service TGS, and a server S. All messages exchanged between these parties are relayed through the client C.

The protocol flow, depicted in Figure B can be divided in three two-message flows, which follow nearly the same pattern: The initiator (always the client C) sends a message to the responder (KDC resp. TGS resp. S, in this order) containing the ID of the client, the ID of the *next* responder, a nonce, and in the last two cases an encrypted authenticator (which contains a timestamp).

The responder answers with two ciphertexts, the one to be decrypted by the initiator containing a nonce, and the one to be forwarded to the next responder containing another timestamp.

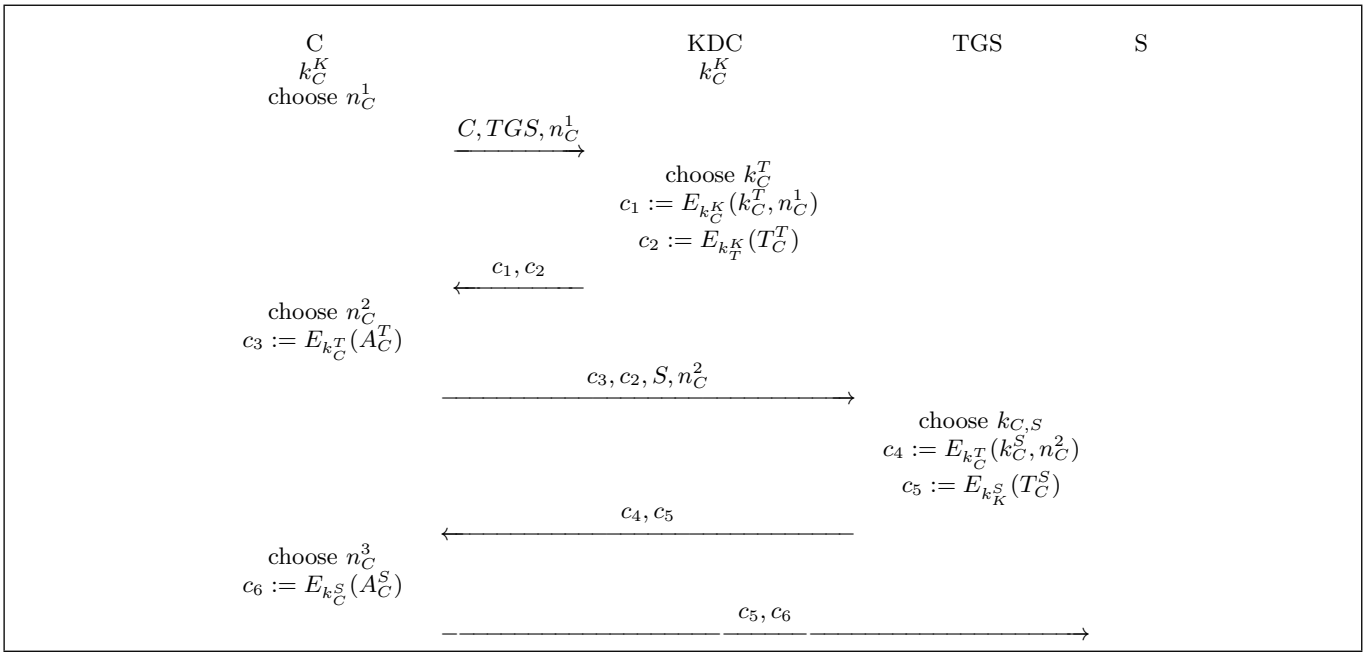

Figure 5: The Kerberos protocol: Overview.