Available online at www.sciencedirect.com
SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 99 (2004) 229–243
www.elsevier.com/locate/entcs

Model-checking the Secure Release of a Time-locked Secret over a Network[1]

Salvatore La Torre[1] Aniello Murano[1,2] Mimmo Parente[1]

[1] *Universtità degli Studi di Salerno, 84041 Baronissi(SA), Italy.*
[2] *Hebrew University, Jerusalem 91904, Israel.*
{slatorre, murano, parente}@unisa.it

Abstract

Weighted timed automata extend timed automata with costs on both locations and transitions. This allows us to associate a quantitative measure to a run, and thus several problems of practical relevance for system design can be studied on this model (optimal reachability, optimal control, etc.). In this paper, we argue that weighted timed automata are suitable models for studying some security properties. In particular, we describe a scenario where we wish to send a message through a network of computers and would like to use sufficient cryptography to ensure the secrecy of the message up to a certain time deadline. We assume that an intruder attempting to decrypt the message can only use resources (computational power) that are local to the nodes visited by the message: we give evidence of this assumption based on a scenario introduced by Rivest, Shamir and Wagner. The computational power of a node is given in terms of number of operations per time unit. We show how this setting can be nicely captured by a weighted timed automaton where: the underlying timed automaton models the features of the network and the weights on the locations represent the computational power of each node. Therefore, we can use model-checking techniques to establish the desired security properties.

Keywords: model-checking, weighted timed automata, timed-release cryptography

1 Introduction

In *Timed-Release Crypto* [12] a message M is encrypted in such a way that nobody (but the *originator*) can decrypt it until a desired amount of time Θ is elapsed. Only after time Θ, the message M is available, and anyone can read it. The novelty of such approach by Rivest et al. is the choice of a

[1] This work was supported by the MURST in the framework of the project “Metodi Formali per la Sicurezza ed il Tempo” (MEFISTO).

doi:10.1016/j.entcs.2004.02.010

symmetric key, that is, the same key is used both to encrypt and to decrypt the message. As usual, the key is a function that is easy to compute in the encryption phase (once the factorization of a large composite number N is known). While in the decryption phase, everybody can decrypt the message only after a pre-determined amount of time Θ has elapsed (unless one succeed in factorizing N). Time Θ can be chosen by the originator of the message as follows: let sq be a basic computer-operation (the squaring operation modulo N) and let t_{sq} be the time required by the destination machine to perform sq, then $\Theta = t_{sq} \cdot k$, where k is a positive integer chosen by the originator. Since this basic operation is intrinsically sequential, time Θ can then be precisely specified for a machine by simply giving the number k of squaring operations to be performed.

As reported in [12] the possible applications of timed-release crypto are many: for example auctions where the bids can be opened after a given deadline, payments that have to be done at the beginning of each month, and so on (see also [6], for other examples).

In this paper we analyze the natural scenario where the timed-release crypto is used on a message that has to get through a network of computers before being delivered at a destination. Thus the sender (Alice) cannot rely only on the speed at which the receiver (Bob) can perform sq as in [12], but has to consider also the traversed nodes of the network that play the adversary role. She aims to put in M as much "difficulty" as needed to achieve the following goals: she wants to be sure that Bob will be able to decipher the message within a time interval pre-determined by her and also, she wishes to keep secret the message contents until a certain time deadline. This scenario is resumed in Figure 1, where the edges are labeled with an interval indicating the minimum and maximum time a message can take to cross that edge and the nodes are labeled with an interval indicating the minimum and maximum time a message can be processed in it before being forwarded to adjacent nodes[2].

Now assume Alice chooses a $k = J^*$, the number of sq operation needed to compute the key, and ciphers M with this key. Let $C_M(k)$ be the ciphered message. She wants to know which are her guarantees injecting the network with $C_M(k)$. More precisely, let $C_M(k)$ be a message ciphered using timed-release crypto such that J^* squaring modulo N operations are needed to decipher it, in this paper we address the following decision problem:

Security Problem: *Given an insecure network of computers G with a source*

[2] Though this information about the network is hard to collect for large networks, as the Internet for example, we can reasonably assume this knowledge for private networks as those used in some corporate.

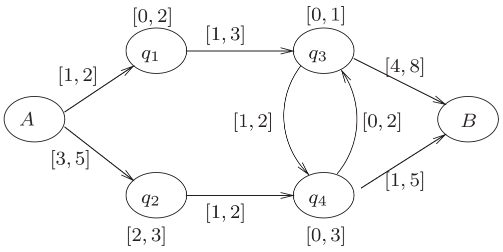

Fig. 1. A network with latency times on the edges and processing times on the nodes.

A and a destination B, a time deadline $\Theta < \Theta_{min}$ and a time interval $[\Theta_{min}, \Theta_{max}]$, does $C_M(k)$ can be sent from A to B such that it cannot be deciphered before time Θ and B can read M at some later time within interval $[\Theta_{min}, \Theta_{max}]$?
We propose to rephrase our security problem within the framework of *weighted timed automata* and then solve it using model-checking techniques.

Timed automata were introduced in [2] to model real-time systems, that is, systems interacting with physical processes and whose correct behavior crucially depends upon real-time considerations. A timed automaton is a finite automaton augmented with a finite set of real-valued *clocks*. Transitions are enabled according to the current *location* and the current clock values. In a transition, clocks can be instantaneously reset, and the value of a clock is exactly the time elapsed since the last time it was reset. Weighted timed automata extend timed automata by associating weights (different costs) with both locations and transitions. This model was independently introduced and studied in [3,5] [3] with the aim of providing timed automata with a performance measure of their finite outcomes. The weight of a location gives the cost of staying at that location per each time unit and the weight of a switch is the cost of taking that discrete transition. Therefore, a cost is associated to each run as the sum of two addend: the sum of the costs of the taken switches and the sum of the costs of the time spent in each location. This way, with each run a linear-cost function of time is associated that accounts for visited locations and taken transitions.

A network can be naturally modeled as a timed automaton where the nodes are represented as locations, and the actions of sending and receiving a message can be modeled as transitions. Time features of the network behaviour (such as traffic delay, latency time, processing time, etc.) can be captured by clocks, resets, and clock constraints. In such a model, weights can be used to capture the computational speed of each node, i.e., the number of squaring operations

[3] In [5] weighted timed automata are named linearly priced timed automata.

per time unit that can be performed at a specific location [4] . In this setting, the security problem can be rephrased as the following decision problem on weighted timed automata. Given a weighted timed automaton $\mathcal{A}$ and a target location T of weight w, for all runs r ending at T of total time $t_r \leq \Theta$ we ask the following questions where J_r denotes the cost of r:

- If $t_r = \Theta$, is J_r less than J^*?
- Is there a $t \geq 0$ such that $t + t_r \in [\Theta_{min}, \Theta_{max}]$ and $w \cdot t + J_r = J^*$?

We refer to this decision problem as the *Time-cost Bounded Reachability problem* (shortly TBR problem). Clearly, to decide this problem it is sufficient to determine the maximum and the minimum cost runs to T among all the runs that reach T within time Θ. In this paper, we give a solution that consists of reducing these problems to analogous problems in a weighted (discrete) directed graph and then uses standard algorithms to solve them. The reduction requires the construction of a graph of exponential size in the length of parameter Θ and in the number of clock variables. If we use Dijkstra's algorithm to compute shortest paths from a single source, to solve the optimization problems on the resulting discrete graph, it turns out that the total time taken by our algorithm is exponential in the length of Θ and in the number of clock variables. For the TBR problem we have a trivial lower bound from reachability in timed automata that is known to be Pspace-complete [2].

We recall that the minimum-cost reachability problem on weighted timed automata was independently studied in [3] and [5]. Let us note that the methods used in those papers do not apply directly to compute the maximum cost of a run. Moreover, also for the computation of the minimum cost of a run, here we consider a simpler setting (we have an upper bound on the time of the runs we consider) and even if this does not allow to achieve dramatic improvements of the computational complexity of the decision algorithms, the description is much simpler. Minimum-cost reachability and minimum-cost control problems on acyclic weighted timed automata were studied in [8]. In [1], a weight w can be associated with each location q such that w gives the cost of a unit of time spent in q. On such a model, given a cost interval I and two states s and t, the decision problem "is t reachable from s at a cost $c \in I$?" (*duration-bounded reachability*) is addressed. Duration-bounded reachability differs from time-cost bounded reachability since the latter has as a parameter also a time bound besides a cost bound. Finally, time-optimal

[4] In a general framework, the computational speed of a node may depend on several parameters: number of performed tasks, resource allocation, and so on. Here, we assume that we have an accurate assessment of the expected values of such parameters and thus of the expected speed at which the squaring operations can be performed at each node of the network.

reachability, that is, the minimum time to reach a given target location, was first considered in [7]. In that paper, the authors addressed the problem of computing lower and upper bounds on time delays in timed automata. Minimum-time reachability is also considered in [11] and the related minimum-time control problem is considered in [4].

The rest of the paper is organized as follows. In Section 2, we recall the main ideas used in timed-release crypto [12] and state our security problem. In Section 3 we recall the definition of weighted timed automata and introduce our notation. Our solution to the TBR problem for weighted timed automata is given in Section 4. Finally, in Section 5 we give a few remarks.

2 Timed-Release Crypto

In this section we recall how the timed-release crypto can be implemented using the so-called "time-lock puzzle" computational problems, following [12]. In Section 2.2 we briefly describe the problem in our setting.

2.1 Time-lock puzzles

A time-lock puzzle is a computational problem (puzzle) that cannot be solved unless a machine performs a computation for at least a certain amount of time. Here is how Rivest et al. have created the time-lock puzzle implementing the timed-release crypto. Alice has a message M for Bob but she wants to encrypt M for a period of time Θ, that is, she does not want Bob (or any other) decrypt the message before time Θ has elapsed. To perform this encryption she executes the following steps:

(i) She generates a composite number N computed as the product of two large primes p and q. Along with it she also computes the Euler function $\Phi(N) = (p-1)(q-1)$.

(ii) She computes $k = \Theta \cdot sp_{Bob}$, where sp_{Bob} is the number of squaring operations modulo N that Bob can perform per time unit (in some sense the speed of Bob's computer).

(iii) She picks a random a modulo N[5] and encrypts M as $C_M(k) = M + a^{2^k} (\bmod N)$. This operation is performed efficiently by computing first $e = 2^k (\bmod \Phi(N))$ and then computing $C_M(k) = M + a^e (\bmod N)$.

(iv) She sends the puzzle $(N, a, k, C_M(k))$ to Bob and erases all variables created (as for p and q).

[5] For practical purposes $a = 2$ can also be used.

The creation and solution of the puzzle is thus based on the computation of the number a^{2^k}, for the chosen k. The scheme used is a symmetric-key encryption/decryption and the key is the number a^{2^k}. If the factorization of N is known, then the number can be computed efficiently by doing $O(\log N)$ multiplications, otherwise the only way to compute a^{2^k} is to compute k iterated squarings, that is, each squaring can only be performed on the result of the preceding squaring. Specifically, one has to compute the function $F(0) = a^{2^0}$ and $F(i) = (F(i-1))^2$, for $0 < i \leq k$. As the squaring is a basic operation, an attempt of parallelization is useless since the communication complexity would be overwhelming, see [10] for more on this.

2.2 Our problem

As described in the previous section, the authors of [12] created a computational problem (a ciphered message $C_M(k)$) that can be solved (deciphered) in a certain amount of time Θ (chosen by Alice, and depending on Bob's computer power to decipher it). Our setting extends their approach in that here $C_M(k)$ has to get through a network before reaching the destination. The computers of the network are all adversaries of Alice playing together: each of them performs some computation on what received, for some time, and then it transmits the result to the adjacent computers. We assume that each node of the network has an interval of time within which it can compute and each link connecting two computers has an interval bounding the time a message can suffer to get through it (both of these intervals represent usual traffic delays in a network), see Figure 1. Obviously, there are other time constraints in the networks, modeling more practical and complicated situations.

In such a scenario, Alice may not be able to choose an instant at which she wants that Bob would decipher the sent message, since there are many computers with possibly different computational capabilities that can be visited each for a different amount of time. Thus, she will rather pick an interval of time $[\Theta_{min}, \Theta_{max}]$ and require that Bob will decipher the message within this interval. Another complication is that Alice cannot refer to a single computer (Bob's computer in the previous scenario) for calculating a suitable number k of squaring modulo N operations that must be performed to ensure decryption within $[\Theta_{min}, \Theta_{max}]$. In fact, she needs to refer to the speed of the computers that $C_M(k)$ may visit but also to the amount of time $C_M(k)$ may spend in each of them. Moreover, for each connecting path it is important to verify its feasibility, that is, whether the path agrees with the time constraints of the network. Therefore, given a network $\mathcal{A}$, instead of the step (ii) of the schema from Section 2.1 Alice will perform the following steps:

1. She chooses a time Θ.
2. She computes the minimum and the maximum number of squaring modulo N operations that the network can perform using exactly time Θ, called J_{min} and J_{max}, respectively.
3. She chooses $J^* > J_{max}$.
4. She computes $\Theta_{min} = \Theta + (J^* - J_{max})/sp_{Bob}$ and $\Theta_{max} = \Theta + (J^* - J_{min})/sp_{Bob}$.

In the rest of the paper, we will discuss a tool for automatically performing the above step 2. Assuming that we can determine J_{min} and J_{max}, directly from the above scheme, the following properties hold.

Proposition 2.1 *For all paths p of $\mathcal{A}$ with finishing time $t_p \leq \Theta$, the total number of squaring operations along p are less than J^*.*

Proposition 2.2 *For all paths p of $\mathcal{A}$ that reach Bob's location at a time $t_p \leq \Theta_{min}$, the residual number of operations to perform can be computed in time t' such that $t' + t_p \in [\Theta_{min}, \Theta_{max}]$.*

Observe that Proposition 2.1 ensures that nobody can decipher the message earlier than time Θ, and Proposition 2.2 ensures that Bob can decipher the message within $[\Theta_{min}, \Theta_{max}]$,

3 Weighted timed automata

In this section, we recall the definition and the notation of timed automata and weighted timed automata.

A timed automaton models a real-time system. We assume that there is a central (real-valued) clock, and the model can use a finite set of *clock variables* (also said simply *clocks*) along with time constraints to check the satisfaction of time requirements. Each clock can be seen as a chronograph synchronized with the central clock, thus it can be read or set to zero (reset): after a reset, a clock restarts automatically. In each automaton, time constraints are expressed by clock constraints. Let C be a set of clocks, the set of clock constraints $\Xi(C)$ contains:

- $x \leq y + c$, $x \geq y + c$, $x \leq c$ and $x \geq c$ $\forall x, y \in C$ and for a natural number c;
- $\neg\delta$ and $\delta_1 \wedge \delta_2$ where $\delta, \delta_1, \delta_2 \in \Xi(C)$.

Furthermore, a *clock interpretation* is a mapping $\nu : C \longrightarrow \mathbb{R}_+$. We denote by $\mathbf{0}$ the clock interpretation mapping each clock to 0. If ν is a clock interpretation, λ is a set of clocks and d is a real number, we denote with $\nu + d$ the clock interpretation that gives the value $\nu(x) + d$ for each clock x, and with

$\lambda(\nu)$ the clock interpretation that differs from ν only on the values of clocks $x \in \lambda$ to which it assigns 0. In particular, we will use $\lambda(\nu + d)$ to denote the clock interpretation that assigns to a clock x value 0, if $x \in \lambda$, and $\nu(x) + d$, otherwise.

Definition 3.1 A *timed automaton* $\mathcal{A}$ is a tuple $(\Sigma, \mathcal{Q}, \mathcal{Q}_0, C, \Delta, inv)$ where:

- Σ is a finite set of symbols (the alphabet);
- $\mathcal{Q}$ is a finite set of locations;
- $\mathcal{Q}_0 \subseteq \mathcal{Q}$ is the set of initial locations;
- C is a finite set of n clock variables;
- Δ is a finite subset of $\mathcal{Q} \times \Sigma \times \Xi(C) \times 2^C \times \mathcal{Q}$ (edges);
- $inv : \mathcal{Q} \longrightarrow \Xi(C)$ maps each location q to its invariant $inv(q)$.

The clock constraint on an edge is called a *guard*. A *state* of a timed automaton $\mathcal{A}$ is a pair $\langle q, \nu\rangle$ where $q \in \mathcal{Q}$ and $\nu \in \mathbb{R}^n_+$. An *initial state* is a pair $\langle q_0, \mathbf{0}\rangle$ where $q_0 \in \mathcal{Q}_0$ is an initial location. The semantics of a timed automaton is given by a transition system over the set of states. The transitions of this system are divided into *discrete steps* and *time steps*. A discrete step is $\langle q, \nu\rangle \xrightarrow{\sigma} \langle q', \nu'\rangle$ where $(q, \sigma, \delta, \lambda, q') \in \Delta$, ν satisfies δ, $\nu' = \lambda(\nu)$, and ν' satisfies $inv(q')$. A time step is $\langle q, \nu\rangle \xrightarrow{d} \langle q, \nu'\rangle$ where $\nu' = \nu + d$, $d \geq 0$, and $\nu + d'$ satisfies $inv(q)$ for all $0 \leq d' \leq d$. A *step* is $\langle q, \nu\rangle \xrightarrow{\sigma,d} \langle q', \nu'\rangle$ where $\langle q, \nu\rangle \xrightarrow{d} \langle q, \nu''\rangle$ and $\langle q, \nu''\rangle \xrightarrow{\sigma} \langle q', \nu'\rangle$, for some $\nu'' \in \mathbb{R}^n$.

A *timed sequence* (σ, τ) over the alphabet Σ is such that $\sigma \in \Sigma^*$, $\tau \in \mathbb{R}^*_+$, and $|\sigma| = |\tau|$.

A run r of a timed automaton $\mathcal{A}$ on a timed sequence (σ, τ), where $\sigma = \sigma_1 \ldots \sigma_k$ and $\tau = \tau_1 \ldots \tau_k$, is a finite sequence $\langle q_0, \nu_0\rangle \xrightarrow{\sigma_1,\tau_1} \langle q_1, \nu_1\rangle \xrightarrow{\sigma_2,\tau_2} \ldots \xrightarrow{\sigma_k,\tau_k} \langle q_k, \nu_k\rangle$. We say that r starts at q_0 and ends at q_k. We denote with $\mathrm{Run}_{\mathcal{A}}$ the set of $\mathcal{A}$ runs, and with $\mathrm{Run}^N_{\mathcal{A}}$ the subset of $\mathrm{Run}_{\mathcal{A}}$ containing all the runs r over a timed sequence (σ, τ) such that τ is a sequence of integers (i.e., delays in these runs are natural numbers). For a run $r = \langle q_0, \nu_0\rangle \xrightarrow{\sigma_1,\tau_1} \langle q_1, \nu_1\rangle \xrightarrow{\sigma_2,\tau_2} \ldots \xrightarrow{\sigma_k,\tau_k} \langle q_k, \nu_k\rangle$, the *signature* of r is $q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_k} q_k$, that is, the sequence of the visited locations and of the taken discrete steps. Moreover, the *total time* taken by r is $\sum_{i=1}^k \tau_i$.

A *weighted timed automaton* is a timed automaton $\mathcal{A}$ with cost functions:

- $J_s : \Delta \longrightarrow \mathbb{N}$ (*switch cost*), and
- $J_d : \mathcal{Q} \longrightarrow \mathbb{N}$ (*duration cost*).

Given a run r of $\mathcal{A}$, let $e_1, \ldots, e_k$ be the sequence of transitions taken in r and $q_0, \ldots, q_k$ be the sequence of visited locations. We associate to r the following costs:

- $J_s(r) = \sum_{i=1}^{k} J_s(e_i)$, and
- $J_d(r) = \sum_{i=0}^{k-1} \tau_i \cdot J_d(q_i)$.

The total cost associated with a run r is then $J(r) = J_s(r) + J_d(r)$.

In the following example we show a weighted timed automaton modeling a simple network of computers.

Example 3.2 Consider the timed transition system in Figure 2. It models as a weighted timed automaton the network from Figure 1. Besides the locations that are already in Figure 1, there are 5 additional locations p_0, p_1, p_2, p_3, p_4 that are used to implement the transmission delays, that is, the time elapsed from the instant at which the message is sent from a node of the network to the instant at which the same message is received at the destination node. We need to use locations to model transmission delays since transitions in timed automata are instantaneous (i.e., they are not time consuming) and we need to distinguish between processing time and transmission time at each node of the network. The only starting location is A. The automaton in Figure 2 uses two clocks. A clock x is used to account for the time spent at each location and thus is reset on each transition (for a clearer representation in Figure 2 we have omitted the resets on the edges) and a clock y is used to measure the total time to reach B from A. Weights on the locations represent the computational power of the corresponding node in the network, that is, the number of operations that can be performed per time unit. Therefore, 0 labels all the locations p_0, p_1, p_2, p_3, p_4 that do not represent an actual node of the network. There are no weights on the edges, and there are only trivial invariants. Therefore, they are omitted in the figure. Guards are used to model processing and transmission delays.

Consider a run of A $r = \langle A, (0,0)\rangle \xrightarrow{0} \langle p_0, (0,0)\rangle \xrightarrow{1.5} \langle q_1, (0,1.5)\rangle \xrightarrow{0.2} \langle p_1, (0,1.7)\rangle \xrightarrow{1} \langle q_3, (0,2.7)\rangle \xrightarrow{0.1} \langle p_3, (0,2.8)\rangle \xrightarrow{6.3} \langle B, (0,9.1)\rangle$. The cost associated with r is $J(r) = 0.2 \cdot w_1 + 0.1 \cdot w_3$ and represents the amount of computation performed in the network on the path from A to B. The total time of r is 9.1. $\square$

4 Decision and optimization problems on WTA

In this section, we discuss two optimization problems on weighted timed automata that address step 2 of the scheme given in Section 2.2. Solving these problems turns out to be sufficient to solve a general decision problem on weighted timed automata that is a natural framework for verifying security

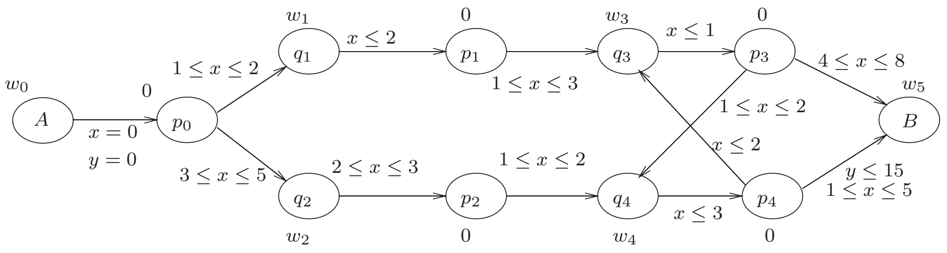

Fig. 2. A simple model of a network.

properties of the type discussed in this paper.

Step 2 of the scheme from Section 2.2 requires the computation of the minimum and the maximum number of squaring operations modulo N that the network can perform using exactly a time Θ chosen by Alice. We can reduce this problem to compute minimum and maximum cost runs in weighted timed automata. For this purpose, we can model the network as a weighted timed automaton as in Example 3.2, then we add a target location T with transitions to T from all locations corresponding to an actual node in the network. We let this added transitions be guarded by $z = \Theta$ where z accounts for the total time since the starting of the computation.

Let $\mathcal{A}$ be a weighted timed automaton and T be a target location. The *maximum-cost reachability at a fixed time* Θ is the problem of computing the maximum cost over all runs ending at T and whose total time is Θ. Analogously, the *minimum-cost reachability at a fixed time* Θ is the problem of computing the minimum cost over all runs ending at T and whose total time is Θ. Let us recall that minimum-cost reachability at a fixed time is a special case of the minimum-cost reachability problems addressed in [3] and [5].

In the following, we will reduce the defined minimum-cost and maximum-cost reachability problems on weighted timed automata to the analogous problems on weighted directed graphs. To simplify the presentation of our results, we restrict to weighted timed automata with guards and invariants that are closed sets (closed guards and invariants). This is without loss of generality, since for a timed automaton $\mathcal{A}$, the minimum and the maximum cost of runs at time Θ are those of the weighted timed automaton $\mathcal{A}'$ that is obtained from $\mathcal{A}$ by changing all the strict inequalities to non-strict ones (see [3]). We start with the following technical lemma that allows us to restrict our search for optimal paths simply to the paths within $\mathrm{Run}_{\mathcal{A}}^N$.

Lemma 4.1 *Let $\mathcal{A}$ be a weighted timed automaton with closed guards and invariants. Given a run $r \in \mathrm{Run}_{\mathcal{A}}$, there are two runs $r_1, r_2 \in \mathrm{Run}_{\mathcal{A}}^N$ such that $J(r_1) \leq J(r) \leq J(r_2)$, and the signature of r_1 and r_2 is the same as the signature of r.*

Proof. Let r be $\langle q_0, \nu_0\rangle \xrightarrow{\sigma_1,\tau_1} \langle q_1, \nu_1\rangle \xrightarrow{\sigma_2,\tau_2} \ldots \xrightarrow{\sigma_k,\tau_k} \langle q_k, \nu_k\rangle$. From the resets, the invariants of the visited locations, and the guards of the taken transitions on r it is possible to construct a system of linear inequalities I over variables $y_1, \ldots, y_k$ such that for every choice of real numbers $d_1, \ldots, d_k$ that satisfy I, $\langle q_0, \nu_0\rangle \xrightarrow{\sigma_1,d_1} \langle q_1, \nu'_1\rangle \xrightarrow{\sigma_2,d_2} \ldots \xrightarrow{\sigma_k,d_k} \langle q_k, \nu'_k\rangle$ is a run of $\mathcal{A}$. Notice that coefficients of variables in I are all 1, and being the constants in the clock constraints natural numbers, we have that all the corner points of the polyhedral set P corresponding to I have natural numbers as components. Recall that a linear cost function over a polyhedral set reaches its minimum/maximum at one of its corner points. Thus, we can pick $(d_1, \ldots, d_k) \in P$ such that the cost of a run $\langle q_0, \nu_0\rangle \xrightarrow{\sigma_1,d_1} \langle q_1, \nu'_1\rangle \xrightarrow{\sigma_2,d_2} \ldots \xrightarrow{\sigma_k,d_k} \langle q_k, \nu'_k\rangle$ is minimized/maximized over P and $d_1, \ldots, d_k$ are integers. Being $(\tau_1, \ldots, \tau_k) \in P$ and since all inequalities in I are not strict, we obtain the lemma. □

The above lemma allows us to compute on a weighted directed graph. In fact, we can restrict to runs that visit states $\langle q, \nu\rangle$ where $\nu(x)$ is an integer for every clock variable x. For the following construction, we assume that $\mathcal{A}$ has a clock that is never reset on all transitions (i.e., we assume that this clock always contains the total time elapsed since the beginning of the computation). Let this clock be y. Let $G_{\mathcal{A}} = (V, E)$ be the weighted graph with weight function W defined as:

- $V = \{\langle q, \nu\rangle \mid \nu(x) \in \{0, \ldots, \Theta\}$ for all $x \in C$ and ν satisfies $inv(q)\}$;
- $(u, v) \in E$ and $W((u, v)) = c$ if and only if either $u \xrightarrow{1} v$ is a time step of $\mathcal{A}$, and $c = J_d(q)$, or $u \xrightarrow{\sigma} v$ is a discrete step of $\mathcal{A}$ corresponding to a transition e and $c = J_s(e)$.

A vertex $\langle q, \nu\rangle$ of $G_{\mathcal{A}}$ such that $\nu(y) = d$ is called a *vertex at time* d. Given a run $r \in \mathrm{Run}^N_{\mathcal{A}}$ such that its total time is not larger than Θ, we denote by $\rho(r)$ the corresponding path of $G_{\mathcal{A}}$. For a given run $r = \langle q_0, \nu_0\rangle \xrightarrow{\sigma_1,\tau_1} \langle q_1, \nu_1\rangle \xrightarrow{\sigma_2,\tau_2} \ldots \xrightarrow{\sigma_k,\tau_k} \langle q_k, \nu_k\rangle$, the path $\rho(r)$ is $u_0 \xrightarrow{w_1} \ldots \xrightarrow{w_m} u_m$ where for $i = 0, \ldots, m$, u_i is $\langle q_j, \nu_j + h\rangle$ with $h \le \tau_{j+1}$, $j \in \{0, \ldots, k\}$ and $i = h + \sum_{l=1}^{j}(\tau_l + 1)$ (assuming that the sum is 0, if $j = 0$)[6]. The total cost associated with the path $\rho(r)$ is $W(\rho(r)) = \sum_{p=1}^{m} w_p$.

The following theorem holds.

Theorem 4.2 *Let $\mathcal{A}$ be a weighted timed automaton, T be a target location, and Θ be a positive real number. It holds that r is a run of $\mathcal{A}$ starting from $q_0 \in \mathcal{Q}_0$, ending at T and whose cost is maximum (resp. minimum) among all the runs of total time Θ if and only if $\rho(r)$ is a maximum-cost (resp.*

[6] The sequence of w_i's is uniquely determined by the sequence of u_i's being $G_{\mathcal{A}}$ a graph.

minimum-cost) path in $G_{\mathcal{A}}$ from $\langle q_0, \mathbf{0}\rangle$ to $\langle T, \nu\rangle$, where $\langle T, \nu\rangle$ is a vertex at time Θ.

Proof. By Lemma 4.1, we can restrict the search for minimum/maximum cost runs to the runs in $\mathrm{Run}_{\mathcal{A}}^{N}$. ¿From the definition of the weighted graph $G_{\mathcal{A}}$ associated to $\mathcal{A}$, we have that a run r of $\mathcal{A}$, starting from $\langle q_0, \mathbf{0}\rangle$ and ending at T if and only if there is a corresponding path $\rho(r)$ in $G_{\mathcal{A}}$ from $\langle q_0, \mathbf{0}\rangle$ to a vertex of the form $\langle T, \nu\rangle$. Moreover, the total time of r is Θ if and only if $\langle T, \nu\rangle$ is a vertex at time Θ. To conclude the proof, we only need to show that the costs of r and $\rho(r)$ coincide.

Let $r = \langle q_0, \nu_0\rangle \xrightarrow{\sigma_1,\tau_1} \langle q_1, \nu_1\rangle \xrightarrow{\sigma_2,\tau_2} \ldots \xrightarrow{\sigma_k,\tau_k} \langle q_k, \nu_k\rangle$. By definition, the path $\rho(r)$ is $u_0 \xrightarrow{w_1} \ldots \xrightarrow{w_m} u_m$ such that each transition $\langle q_j, \nu_j\rangle \xrightarrow{\sigma_{j+1},\tau_{j+1}} \langle q_{j+1}, \nu_{j+1}\rangle$ in r corresponds to a sub-path $u_i \xrightarrow{w_{i+1}} \ldots \xrightarrow{w_{i'}} u_{i'}$ in $\rho(r)$, where $i = \sum_{l=1}^{j}(\tau_l + 1)$ (if $j = 0$, then $i = 0$) and $i' = i + (\tau_{j+1} + 1)$. Recall that for $0 \le p \le \tau_{j+1}$, $u_{i+p} = \langle q_j, \nu_j + p\rangle$, and $W((u_{i+p}, u_{i+p+1})) = J_d(q_j)$. Moreover, $u_{i'-1} = \langle q_j, \nu_j + \tau_{j+1}\rangle$ and $W((u_{i'-1}, u_{i'})) = J_s(e_j)$ where e_j is the discrete step $\langle q_j, \nu_j + \tau_{j+1}\rangle \xrightarrow{\sigma_{j+1}} \langle q_{j+1}, \nu_{j+1}\rangle$. Thus, the cost associated with $u_i \xrightarrow{w_{i+1}} \ldots \xrightarrow{w_{i'}} u_{i'}$ is $\tau_{j+1} \cdot J_d(q_j) + J_s(e_j)$, that is exactly the cost associated with $\langle q_j, \nu_j\rangle \xrightarrow{\sigma_{j+1},\tau_{j+1}} \langle q_{j+1}, \nu_{j+1}\rangle$.

Iterating this argument for every $j = 0, \ldots k$, we have that $J(r) = W(\rho(r))$. □

Using the above results we have the following theorem.

Theorem 4.3 *Minimum-cost (resp. maximum-cost) reachability at a fixed time for weighted timed automata can be solved in exponential time.*

Proof. By Theorem 4.2, given a weighted timed automaton $\mathcal{A}$, we can restrict such problems to search for minimum and maximum-cost paths in the weighted directed graph $G_{\mathcal{A}}$. By a simple counting the number of vertices of $G_{\mathcal{A}}$ is $|\mathcal{Q}| \cdot (\Theta + 1)^n$, where $\mathcal{Q}$ is the set of locations and n is the number of clocks of $\mathcal{A}$. Thus, the size of $G_{\mathcal{A}}$ is linear in the number of locations and exponential in both the length of the representation of Θ and the number of clocks. To compute optimal weighted paths on a weighted directed graph, we can use a standard algorithm that runs in polynomial time in the number of vertices. For example, we could use Dijkstra's algorithm to compute the shortest paths from a single source. Hence, we get the theorem. □

We end this section discussing a decision problem related to the considered optimization and security problems.

Let $\Theta \le \Theta_{min} \le \Theta_{max}$ and J^* be natural numbers. Given a weighted timed automaton $\mathcal{A}$ and a target location T of weight w, we are interested in the following decision problems:

(**Problem** 1) For all the runs r ending at T whose total time is at most Θ, we wish to determine if $J(r)$ is less than J^*.

(**Problem** 2) For all the runs r ending at T whose total time is Θ, we wish to determine if there exists a $t \geq 0$ such that $t + \Theta \in [\Theta_{min}, \Theta_{max}]$ and $t \cdot w + J(r) = J^*$.

We refer to the whole of Problems 1 and 2 above as the *Time-cost Bounded Reachability* problem. To solve Problem 1, we can just compute the maximum cost of a run to T within time Θ. While to solve Problem 2, besides solving the maximum-cost reachability problem we also compute the minimum cost of a run to T at time Θ. In fact, let J_{min} and J_{max} be respectively the minimum cost and the maximum cost of a run to T at time Θ. Clearly, there exists a t satisfying the condition required by Problem 2 if and only if both $J_{min} + (\Theta_{max} - \Theta) \cdot w \geq J^*$ and $J_{max} + (\Theta_{min} - \Theta) \cdot w \leq J^*$ hold.

Using the above results we obtain the desired result.

Theorem 4.4 *Time-cost bounded reachability for weighted timed automata is decidable and can be solved in exponential time.*

Since the reachability problem in timed automata turns out to be PSPACE-complete [2] we also get the following lower bound for the problem.

Corollary 4.5 *Time-cost bounded reachability problem for weighted timed automata is* PSPACE-*hard.*

5 Conclusions

In this paper, we have dealt with timed-release cryptography across a network of computers. Timed-release crypto was introduced and studied by Rivest at al. in [12], motivated by the idea of sending "information to the future". In this paper, we have extended their approach to analyze a natural scenario where the timed-release crypto is used on a message that has to get through a network of computers before being delivered at a destination. Thus, several parameters of the used network need to be taken in account in order to ensure the secrecy of the message up to a certain time deadline. In particular, for a given insecure network of computers G with source A and destination B, we have addressed the following security problem: "Does a ciphered message $C_M(k)$ can be sent from A to B such that it cannot be deciphered before a fixed time deadline and can instead be deciphered at B within a time interval $[\Theta_{min}, \Theta_{max}]$?"

We have proposed a solution of this security problem based on weighted timed automata. The solution consists of rephrasing the problem into a decision problem for such class of automata, the time-cost bounded reachability. We have studied this problem and proposed an algorithm that takes time exponential in the size of the time deadline and in the number of clocks used in the automaton. We observe that it is unlikely that this problem can admit a sub-exponential solution, since from the computational complexity of reachability in standard timed automata [2], we get that this problem is Pspace-hard.

In some situations, it is reasonable to assume that the receiver of the message can control only some nodes of the network and thus routes the message only to these adjacent nodes instead of to others. We could capture this framework by modeling the network as a two-player game on a weighted timed automaton, and search for a winning strategy of the receiver. We will explore this scenario in a future research.

References

[1] R. Alur, C. Courcoubetis, and T.A. Henzinger. Computing accumulated delays in real-time system. In *Proc. of the Fifth International Conference on Computer-Aided Verification, CAV'93*, LNCS 697, pages 181 – 193. Springer, 1993.

[2] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183 – 235, 1994.

[3] R. Alur, S. La Torre, and G. J. Pappas. Optimal paths in weighted timed automata. In *Proc. of the 4th International Workshop on Hybrid Systems: Computation and Control, HSCC'01*, LNCS 2034, pages 49 – 62. Springer, 2001. *To appear in Theoretical Computer Science*.

[4] E. Asarin and O. Maler. As soon as possible: Time optimal control for timed automata. In *Proc. of the 2nd International Workshop on Hybrid Systems: Computation and Control*, LNCS 1569, pages 19 – 30. Springer, 1999.

[5] G. Behrman, T. Hune, A. Fehnker, K. Larsen, P. Pettersson, R. Romijn, and F. Vaandrager. Minimum-cost reachability for priced timed automata. In *Proc. of the 4th International Workshop on Hybrid Systems: Computation and Control, HSCC'01*, LNCS 2034, pages 147–161. Springer, 2001.

[6] D. Boneh, M. Naor, Timed Commitments, Advances in Cryptology: Proceedings of CRYPTO '00, Lecture Notes in Computer Science 1880, Springer-Verlag 2000, 236–254.

[7] C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. In *Proc. of the 3rd International Conference on Computer Aided Verification*, LNCS 575, pages 399 – 409. Springer, 1991.

[8] S. La Torre, S. Mukhopadhyay, and A. Murano. Optimal-Reachability and Control for Acyclic Weighted Timed Automata. In *2nd IFIP International Conference on Theoretical Computer Science, IFIP TCS'02*, pages 498 –510. Kluwer, 2002.

[9] K. G. Larsen, G. Behrman, E. Brinksma, A. Fehnker, T. Hune, P. Petersson, and J. Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In *Proc. of the 13th International Conference on Computer Aided Verification, CAV'01*, LNCS, pages 493–505. Springer, 2001.

[10] W. Mao, Timed-Release Cryptography, In Proceedings of SAC 2001, S. Vaudenay and A. Youssef (Eds.), Lecture Notes in Computer Science Vol. 2259, 342–357, 2001.

[11] P. Niebert, S. Tripakis, and S. Yovine. Minimum-time reachability for timed automata. In *Proc. of the 8-th IEEE Mediterranean Conference on Control and Automation*, 2000.

[12] R.L. Rivest, A. Shamir, D. A. Wagner, Time-Lock puzzles and Timed-Release Crypto, Technical report, MIT/LCS/TR-684. Available at the URL: (theory.lcs.mit.edu/ rivest/RivestShamirWagner-timelock.ps).