(2) $\forall\ i \in \mathbb{N}^+,\ c \notin R(i) \implies H_R^c(i+1) = H_R^c(i)$; *(3)* $\forall\ i \in \mathbb{N}^+,\ c \in R(i) \implies H_R^c(i+1) = H_R^c(i)\ +\ 1$.

A probabilistic *relation* in PrCcsl is satisfied if and only if the probability of the *relation* constraint being satisfied is greater than or equal to the probability threshold $p \in [0,\ 1]$. Given k `runs` $= \{R_1, \ldots, R_k\}$, the probabilistic `subclock`, `coincidence`, `exclusion` and `precedence` in PrCcsl are defined as follows:

Probabilistic Subclock: $c1 \subseteq_p c2 \iff Pr[c1 \subseteq c2] \geqslant p$, where $Pr[c1 \subseteq c2] = \frac{1}{k} \sum_{j=1}^{k} \{R_j \models c1 \subseteq c2\}$, representing the ratio of `runs` that satisfies the relation out of k `runs`. A `run` R_j satisfies the `subclock` relation between $c1$ and $c2$ "if $c1$ ticks, $c2$ must tick" holds at every step i in R_j, s.t., $(R_j \models c1 \subseteq c2) \iff (\forall i\ 0 \leqslant i \leqslant n,\ c1 \in R(i) \implies c2 \in R(i))$. "$R_j \models c1 \subseteq c2$" returns 1 if R_j satisfies $c1 \subseteq c2$, otherwise it returns 0.

Probabilistic Coincidence: $c1 \equiv_p c2 \iff Pr[c1 \equiv c2] \geqslant p$, where $Pr[c1 \equiv c2] = \frac{1}{k} \sum_{j=1}^{k} \{R_j \models c1 \equiv c2\}$, which represents the ratio of `runs` that satisfies the `coincidence` relation out of k `runs`. A `run`, R_j satisfies the `coincidence` relation on $c1$ and $c2$ if the assertion holds: $\forall i,\ 0 \leqslant i \leqslant n,\ (c1 \in R(i) \implies c2 \in R(i)) \wedge (c2 \in R(i) \implies c1 \in R(i))$. In other words, the satisfaction of `coincidence` relation is established when the two conditions "if $c1$ ticks, $c2$ must tick" and "if $c2$ ticks, $c1$ must tick" hold at every step.

Probabilistic Exclusion: $c1 \#_p c2 \iff Pr[c1 \# c2] \geqslant p$, where $Pr[c1 \# c2] = \frac{1}{k} \sum_{j=1}^{k} \{R_j \models c1 \# c2\}$, indicating the ratio of `runs` that satisfies the `exclusion` relation out of k `runs`. A `run`, R_j, satisfies the `exclusion` relation on $c1$ and $c2$ if $\forall i,\ 0 \leqslant i \leqslant n,\ (c1 \in R(i) \implies c2 \notin R(i)) \wedge\ (c2 \in R(i) \implies c1 \notin R(i))$, i.e., for every step, if $c1$ ticks, $c2$ must not tick and vice versa.

Probabilistic Precedence: $c1 \prec_p c2 \iff Pr[c1 \prec c2] \geqslant p$, where $Pr[c1 \prec c2] = \frac{1}{k} \sum_{j=1}^{k} \{R_j \models c1 \prec c2\}$, which denotes the ratio of `runs` that satisfies the `precedence` relation out of k `runs`. A `run` R_j satisfies the `precedence` relation if the condition $\forall i,\ 0 \leqslant i \leqslant n,\ (H_R^{c1}(i) \geqslant H_R^{c2}(i))$ and $(H_R^{c2}(i) = H_R^{c1}(i)) \implies (c2 \notin R(i))$ hold, i.e., the history of $c1$ is greater than or equal to the history of $c2$, and $c2$ must not tick when the history of the two clocks are equal.

2.2 UPPAAL-SMC

UPPAAL-SMC [31] performs the probabilistic analysis of properties by monitoring simulations of the complex hybrid system in a given stochastic environment and using results from the statistics to determine whether the system satisfies the property with some degree of confidence. Uppaal-SMC provides a number of queries related to the stochastic interpretation of Timed Automata (STA)

[8] and they are as follows, where N and *bound* indicate the number of simulations to be performed and the time bound on the simulations respectively: 1. *Probability Estimation* estimates the probability of a requirement property ϕ being satisfied for a given STA model within the time bound: $Pr[bound]\ \phi$; 2. *Hypothesis Testing* checks if the probability of ϕ is satisfied within a certain probability P_0: $Pr[bound]\ \phi \geq P_0$; 3. *Simulations*: UPPAAL-SMC runs multiple simulations on the STA model and the k (state-based) properties/expressions $\phi_1, ..., \phi_k$ are monitored and visualized along the simulations: *simulate* N $[\leq bound]\{\phi_1, ..., \phi_k\}$.

3 Running Example

A cooperative automotive system (CAS) [13] is adopted to illustrate our approaches. CAS includes distributed and coordinated sensing, control, and actuation over three vehicles (denoted as v_i, where $i \in \{0, 1, 2\}$) which are running in the same lane. As shown in Fig. 1, a lead vehicle (v_0) runs automatically by recognizing traffic signs on the road. The following vehicle must set its desired velocity identical to that of its immediate preceding vehicle. Vehicles should maintain sufficient braking distance to avoid rear-end collision while remaining close enough to guarantee communication quality. Vehicle movement relies on availability of environmental information, e.g., traffic signs, obstacles, etc. The position of v_i is represented by Cartesian coordinate (x_i, y_i), where x_i and y_i are distances measured from the vehicle to the two fixed perpendicular lines, i.e., x-axis and y-axis, respectively.

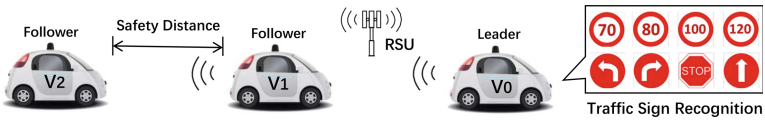


Fig. 1. Overview of Cooperative Automotive System

The cooperative driving of CAS requires prompt and secure information transmission among vehicles. We adopt a roadside unit aided (RAISE) [33] communication protocol in VANET to achieve the data transmission. Each vehicle periodically broadcasts its own position and velocity to its immediate following vehicle through wireless connection. The authentication of the identities of each vehicle and verification of messages sent by the vehicles is performed by RSU. For further details of RAISE, refer to Sect. 4.1. The following S/S properties on CAS are considered:

R1. The follower vehicle should not overtake its leading vehicle when the vehicles run at a positive direction of x-axis.

R2. When the lead vehicle detects a stop sign, all the three vehicles must stop within a given time, e.g., 2000 ms.

R3. If the distance between a vehicle and its preceding vehicle is less than minimum safety distance, the vehicle should decelerate within a certain time (200 ms).
R4. If the distance between a vehicle and its preceding vehicle is greater than the maximum safety distance (e.g., 100 m), the vehicle should accelerate within a certain time, e.g., 300 ms.
R5. When the lead vehicle starts to turn left (or turn right), the two follower vehicles should finish turning and run in the same lane within a given time.
R6. Authenticity: If a vehicle receives a message, its preceding vehicle must have sent a corresponding message before, i.e., the protocol should be resistant to message spoofing attack.
R7. Secrecy: Symmetric keys of vehicles should be kept confidential to attackers.
R8. Integrity: The content of messages must not be modified during transmission, i.e., the protocol should be resistant to message falsification attack.
R9. Freshness: The vehicles should not accept an "obsolete" message, namely, the difference between the current time and the *timestamp* of the accepted message should be less than the predefined time threshold.
R10. The symmetric key agreement (i.e., mutual authentication) process between RSU and three vehicles should be completed within a certain time, e.g., 600 ms.
R11. A vehicle should send messages to its subsequent vehicle periodically with a period 200 ms and a jitter 100 ms.

Among the above S/S requirements, R1–R5 are safety [20] properties, which specify that the system should not cause undesirable results on its environment and aim at protecting human lives, health and assets from being damaged. R6-R11 are security properties, which refer to the inability of the environment to affect the system in an undesirable way and aim to guarantee the confidentiality and integrity of transmitted information. The interdependencies among those S/S properties are conditional dependencies [17], i.e., violations of security properties can lead to the violations on safety properties. The events associated with those S/S properties can be interpreted as logical clocks in PrCcsl, which provides a way to express S/S properties in the logical time manner [16]. Therefore, S/S properties can be interpreted as logical timing constraints, i.e., the temporal and causality clock *relations* in PrCcsl.

The methodology for analysis of S/S related timing constraints in this paper can be generalized in Fig. 2. First, on the basis of the existing behavioral model of CAS described in [13], we enhance the CAS model by augmenting (parallelly composing) it with models of Raise protocol and malicious attacks, resulting in a refined CAS model regarding vehicular communication characteristics and security-related adversary interference. Second, we specify S/S timing constraints (R1–R11) in PrCcsl and translate the PrCcsl specifications into corresponding STA and probabilistic queries. Finally, we combine the model of CAS and the STA of PrCcsl specifications, and perform formal verification based on the combined model using Uppaal-SMC.

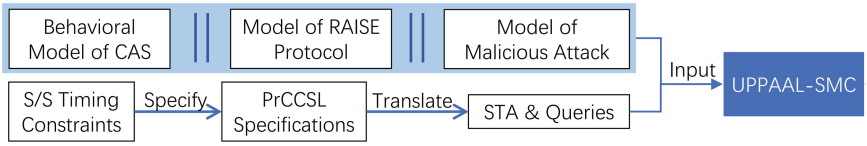


Fig. 2. Methodology for analysis of S/S timing constraints

4 Modeling and Refinement of CAS in UPPAAL-SMC

The behaviors of CAS are modeled as a network of stochastic timed automata (NSTA) in Uppaal-SMC described in [13]. In this section, we refine the CAS model by adding it with the models of Raise protocol and security attacks.

4.1 Modeling of RAISE Protocol in UPPAAL-SMC

We present a simplified version of Raise protocol [33] and its Uppaal-SMC model. The original Raise protocol is modified to facilitate the communication mechanism of CAS, i.e., each follower vehicle receives messages from its immediate preceding vehicle and RSU. Furthermore, timing constraints are also appended to restrict the time duration of each step (e.g., encryption and decryption) during communication process. There are two phases in Raise protocol, i.e., *symmetric key agreement* and *information transmission*.

1. **Symmetric key agreement (SKA)** is performed to obtain symmetric key k_i for guaranteeing security of communication and generates pseudo identities ID_i of vehicles for covering their real identities. The shared symmetric key between RSU and v_i is $k_i = g^{ab}$, where g, a, b are three positive random numbers. As shown in Fig. 3, $Encry(msg, k)$ ($Decry(msg, k)$) denotes the encryption (decryption) of message msg with key k, where k can be either a public key or symmetric key. $Sign(msg, k)$ generates signature of msg with a private key k. We use PK_i to denote the public key of v_i and SK_i to represent the corresponding private key. "||" is the concatenation operation on messages.

Initially, v_i randomly picks g and a (step 1), encrypts "$g||a$" and sends the encrypted result (m_i) to RSU (step 2). Upon receiving m_i, RSU decrypts the message (step 3). It then generates b and ID_i, signs and sends the signed message (rm_i) to v_i (step 4 and 5). v_i verifies the rm_i's signature (step 6) and sends back the signature of $g||a||b||ID_i$ (step 7). Finally, RSU verifies the signature s_i (step 8). If all the steps are completed correctly, the key agreement process succeeds.

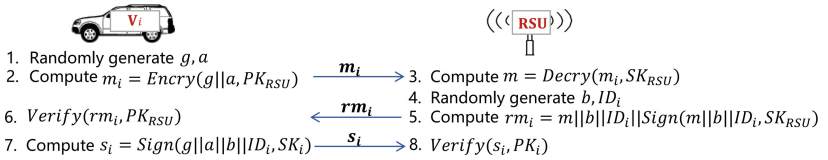


Fig. 3. Symmetric key agreement in Raise

2. **Information transmission (IT)** initiates after the SKA is completed. The traffic information (i.e, brake, direction, position and speed) of v_i is integrated into a message $msg_i = brake_i||direction_i||x_i||y_i||speed_i$. As presented in Fig. 4, initially, v_i generates the message authentication code (MAC) of msg_i with the symmetric key k_i (generated in SKA). Then, v_i concatenates the MAC code with

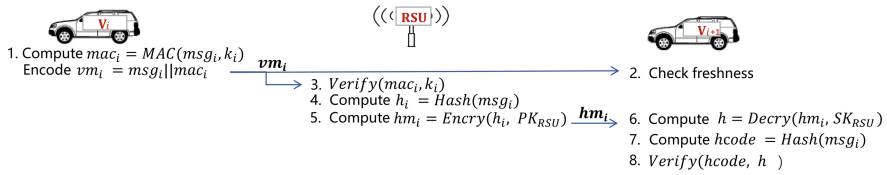


Fig. 4. Information transmission in RAISE

msg_i and sends it to RSU and v_{i+1} (step 1). Upon receiving vm_i, v_{i+1} checks the freshness of the message (step 2), i.e., if the time interval between the current time and the time when vm_i is sent is greater than the predefined threshold, v_{i+1} drops vm_i. At the same time, RSU checks the authenticity of vm_i (step 3). If mac_i is correct, RSU computes the hash code h_i of message msg_i (step 4). Afterwards, it encrypts h_i and sends the encrypted result hm_i to v_{i+1} (step 5). v_{i+1} decrypts hm_i and get the hash code h (step 6). Furthermore, to ensure the consistency of the message, v_{i+1} itself also computes the hash code of msg_i (step 7). It then verifies whether the hash code calculated by itself is the same as the decrypted hash code and decides to accept or reject msg_i (step 8).

To model RAISE in UPPAAL-SMC, interactions among vehicles and RSU (i.e., sending/receiving messages) are modeled by *synchronization channels* [31] and global variables. The cryptographic operations in RAISE refer to public and private key encryption and decryption, i.e., a message encrypted by public key can be decrypted using the corresponding private key, and vice versa. The automaton of `cryptographic device` [6] is adopted to model the encryption and decryption. Figure 5 presents the STA capturing behaviors of vehicle v_i and RSU in SKA. *startEn* (resp. *startDe*) and *finDe* (resp. *finEn*) are channels for indicating the starting and finishing of encryption (resp. decryption). The encryption/decryption result is denoted *en_res*/*de_res*. In the STA, names of locations indicate the corresponding steps pictured in Fig. 3.

IT phase from v_0 to v_1 is established with the help of RSU, modeled as the STA shown in Fig. 6 (the transmission from v_1 to v_2 can be modeled similarly). The behaviors of v_0 (sender), v_1 (receiver) and RSU in the IT phase are modeled in `IT_v0`, `IT_v1` and `IT_RSU` STA, respectively.

The SKA (or IT) succeeds if each step of the SKA (IT) is completed correctly within a given time interval, modeled by invariant "t $\leq$ d" (the value of d varies in different steps). If timeout occurs (i.e., "t $\geq$ d"), *fail* location will be activated and the procedure is restarted from the initial step.

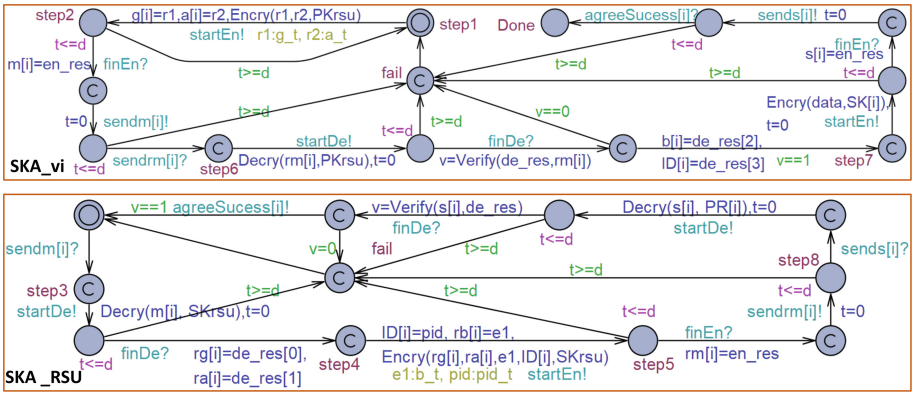


Fig. 5. UPPAAL-SMC model of SKA

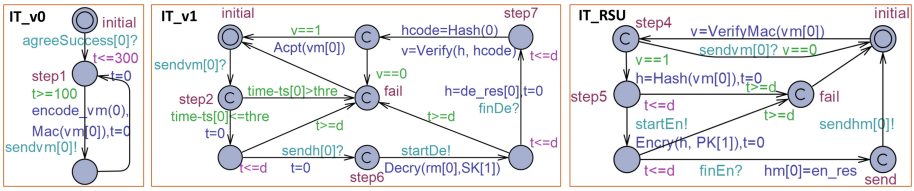


Fig. 6. UPPAAL-SMC model of IT

4.2 Modeling of Attacks in UPPAAL-SMC

We present the modeling of three types of attacks commonly used in the security analysis, i.e., message falsification, message replaying and message spoofing attacks [2]. The models of attacks are illustrated in Fig. 7, where the *ls* parameter ($ls \in [0, 100]$) serves as an indicator of level of adversarial strength while *qc* ($qc \in [0, 100]$) is an indicator of the adversarial channel quality.

Message Falsification Attack (MFA) aims to falsify messages transmitted from v_i to v_{i+1}, which is modeled as `MFA` STA in Fig. 7. As described earlier, in RAISE, RSU verifies the authenticity of messages by checking the correctness of the MAC code of messages. To deceive the RSU on the validity of the modified message and avoid exposing itself to RSU, MFA attempts to obtain the symmetric key and utilizes the key to compute the MAC code of the falsified message. At `s1` state, `MFA` eavesdrops on rm_i (generated at step 5 in Fig. 3), which contains the information for symmetric key generation (i.e., *g*, *a*, *b*). It tries to decrypt rm_i when receiving it via $sendrm[i]?$. The probability that the decryption can succeed is $ls\%$, modeled by probabilistic choices [31] (dashed edges) with probability weight as $\frac{ls}{100}$ and $\frac{100-ls}{100}$. If the decryption succeeds, MFA obtains the symmetric key of v_i based on the decrypted result ($getKey(de\_res)$). Finally, it modifies the content of message using the key, and tries to send the modified message to v_{i+1} ($sendvm[i]!$). The probability that the message can be sent successfully is

(100-qc)%. In our setting, MFA modifies the $speed_i$ field in the message into a random value in [100, 120], and changes the direction as $direction_i = 4$, which indicates that the v_i is running at the positive direction on y-axis.

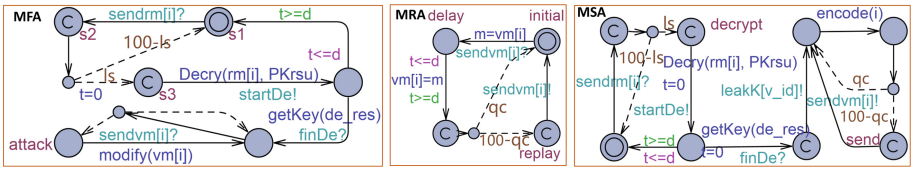


Fig. 7. STA of attacks

Message Replaying Attack (MRA) targets to replay obsolete messages that contain old information. The `MRA` STA represents an MRA that replays messages sent by v_i. Upon capturing a message (via $sendvm[i]?$), MRA stores the message ($m = vm[i]$) and tries to replay it at a later time (i.e., after *10* s). The probability that the attacker can replay the message successfully is (100-qc)%.

Message Spoofing Attack (MSA) impersonates a vehicle (v_i) in order to inject fraudulent information into its subsequent vehicle (v_{i+1}). Similar to `MFA`, `MSA` STA first obtains the symmetric key of v_i by detecting and decrypting rm_i. It then fabricates a new message whose content is "$brake_i = 0$, $speed_i = 0$, $direction_i = 4$, $x_i = 0$, $y_i = 10$" (denoted "*encode(i)*") and tries to send the message to v_{i+1} ($sendvm[i]!$), with the probability of the message being sent successfully as (100-qc)%.

5 Representation of S/S Related Timing Constraints in UPPAAL-SMC

To enable the formal verification of S/S related timing constraints (given in Sect. 3), we first investigate how to specify those constraints in PrCcsl. Then, translation from PrCcsl specifications of the constraints into verifiable STA is demonstrated. Furthermore, a tool ProTL that supports the automatic transformation based on the proposed translation rules is introduced.

5.1 Specifications of S/S Related Timing Constraints in PrCCSL

The specifications of R1–R11 are presented in Table 1, where *ac* is a clock that always ticks while *nc* represents a clock that never ticks. R1 is specified as an `exclusion` *relation* between *xdir* (the event that the vehicles are running at the positive direction of x-axis) and *ovtake* (the event that the position of follower v_1 on x-axis is greater than that of leader v_0). Similarly, R7 and R9 can be specified as `exclusion` *relations*.

In the specification of R2, *stopD* is a clock generated by delaying *stopSign* (the event that the leader vehicle detects a stop sign) for 2000 ms. *vstop* refers

Table 1. PrCCSL specifications of R1–R11

Req	PrCCSL Specification
R1	$xdir \triangleq dir = 1?\ ac : nc$, $ovtake \triangleq x_1 \geq x_0\ ?\ ac : nc$, $xdir\ \#_{0.95}\ ovtake$
R2	$stopSign \triangleq sign = 5\ ?\ signRec : nc$, $stopD \triangleq stopSign\ (2000) \rightsquigarrow ms$, $vstop \preceq_{0.95} stopD$
R3	$vUnsafeDe \triangleq vUnsafe\ (200) \rightsquigarrow ms$, $vDec \prec_{0.95} vUnsafeDe$
R4	$vFarDisDe \triangleq vFarDis\ (300) \rightsquigarrow ms$, $startAcc \prec_{0.95} vFarDisDe$
R5	$v0TurnDe \triangleq v0Turn\ (3000) \rightsquigarrow ms$, $finTurn \preceq_{0.95} v0TurnDe$
R6	$msgRec \subseteq_{0.95} msgSent$
R7	$leakK\ \#_{0.95}\ ac$
R8	$validMsg \triangleq rMsg = sMsg\ ?\ msgRec : nc$, $msgRec \equiv_{0.95} validMsg$
R9	$oldMsg \triangleq time - ts > thre\ ?\ msgAcpt : nc$, $msgAcpt\ \#_{0.95}\ oldMsg$
R10	$startSKADe \triangleq startSKA\ (600) \rightsquigarrow ms$, $finSKA \prec_{0.95} startSKADe$
R11	$fclk \triangleq msgSent \blacktriangledown 01(1)$, $sentDe1 \triangleq msgSent\ (100) \rightsquigarrow ms$, $sentDe2 \triangleq msgSent\ (300) \rightsquigarrow ms$, $sentDe1 \preceq_{0.95} fclk$, $fclk \preceq_{0.95} sentDe2$

to the event that three vehicles are completely stopped, which should occur no later than *stopD*. Hence, R2 is expressed as a `causality` *relation* between *vstop* and *stopD*. R3–R5 can be specified in a similar manner.

R6 (authenticity) is expressed as a `subclock` *relation* between *msgRec* and *msgSent*, where *msgRec* (*msgSent*) represents the event that a message is received (sent) by the follower (leader) vehicle. R8 is specified as a `coincidence` *relation* between *msgRec* and *validMsg*, where *validMsg* is a clock that ticks with *msgRec* when the received message *rMsg* is identical with the sent message *sMsg* (i.e., $rMsg == sMsg$). For R10, *startSKA* (*finSKA*) represents the starting (completion) of SKA. *startSKADe* is a clock constructed by delaying *startSKA* for 600 ms. R10 delimits that *finSKA* must occur before *startSKADe*. R11 states that two consecutive occurrences of *msgSent* must has a interval of $[period - jitter,\ period + jitter]$ms (i.e., [100, 300] ms). In the specification of R11, *fclk* is a clock generated by filtering out the 1^{st} tick of *msgSent*. *sentDe1* and *sentDe2* are two clocks generated by delaying *msgSent* for 100 ms and 300 ms. R11 can be interpreted as: $\forall i \in \mathbb{N}^+$, the i^{th} tick of *fclk* should occur later than the i^{th} tick of *sentDe1* but prior to the i^{th} tick of *sentDe2*.

5.2 Translation of PrCCSL into STA

We present how the S/S related timing constraints specified in PrCcsl can be transformed into STA and probabilistic queries in Uppaal-SMC. We first describe how clock tick and history (introduced in Sect. 2) can be represented in Uppaal-SMC. Using the mapping, we then demonstrate that *expressions* and *relations* in PrCcsl can be translated into STA and queries.

In the earlier work [14], the semantics of PrCCSL operators are translated into STA based on discrete time, i.e., the continuous physical time is discretized into a set of equalized steps. As a result, two clock instants are still considered coincident even if they are one time step apart. To alleviate this restriction and enable the representation of PrCCSL that pertains to continuous real-time semantics, the mapping patterns are refined: two clock instants are coinstantaneous only if the time difference between them is insignificant, i.e., the time difference between them is less than a positive infinitesimal value e, e.g., $e = 0.000001$.

In PrCCSL, a logical clock represents an event and the instants of the clock correspond to the occurrences of the event. A logical clock c is represented as a *synchronization channel* $c!$ in UPPAAL-SMC. The history of c is modeled as the STA shown in Fig. 8: whenever c occurs ($c?$), the value of its history is increased by 1 (i.e., $h{+}{+}$).

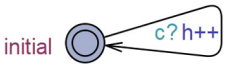


Fig. 8. `History`

Based on the mapping patterns of tick and history, the PrCCSL *expressions* (including `ITE`, `DelayFor` and `filterBy`), as well as *relations* (including `subclock`, `coincidence`, `exclusion` and `precedence`), can be represented as STA and queries shown in Fig. 9.

The STA of *expressions* trigger the ticks of the new clock (denoted *res*!) based on the occurrences of existed clocks. To represent *relations*, observer STA that capture the semantics of standard `subclock`, `coincidence`, `exclusion` and `precedence` *relations* are constructed. Each observer STA contains a "*fail*" location (see Fig. 9), which indicates the violation of the corresponding *relation*. Recall the definition of PrCCSL in Sect. 2, the probability of a *relation* being satisfied is interpreted as a ratio of runs that satisfies the *relation* among all runs. It is specified as *Hypothesis Testing* queries in UPPAAL-SMC, H_0: $\frac{m}{k} \geqslant p$ against H_1: $\frac{m}{k} < p$, where m is the number of runs satisfying the given *relation* out of all k runs. As a result, the probabilistic *relations* are interpreted as the query (see Fig. 9): $Pr[bound]([\,]\ \neg STA.fail) \geq p$, which means that the probability of the "*fail*" location of the observer STA never being reached should be greater than or equal to p. The STA of *expressions* and *relations* are composed to the system NSTA in parallel. Then, the probabilistic analysis is performed over the composite NSTA that enables us to verify the S/S related timing constraints over the entire system using UPPAAL-SMC.

Tool support: Manual translation of PrCCSL specifications into UPPAAL models for verification can be time-consuming and error-prone. To improve the accuracy and efficiency of translation, we implement a tool ProTL (Probabilistic-CCSL TransLator) [26] that provides a push-button transformation from PrCCSL specifications into corresponding STA & queries. Furthermore, verification and simulation support is provided in ProTL by employing the UPPAAL-SMC as the backend analysis engine. ProTL encompasses the following features: (1) An editor for editing PrCCSL specification of requirements (stored as *".txt"* files); (2) Automated transformation of PrCCSL specifications into UPPAAL-SMC STA; (3) Integration of the STA and the system behavioral model (imported by users); (4) A configuration palette for setting parameters (e.g., time bound of simula-

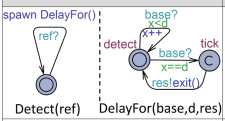
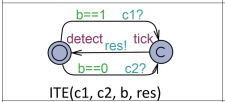
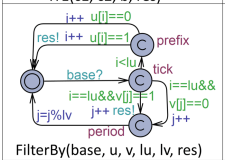
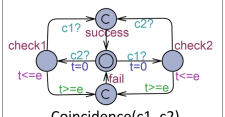
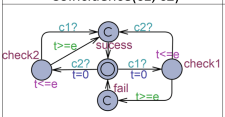
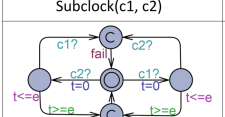
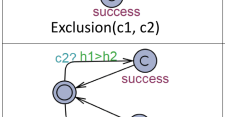
STA of PrCCSL Operators	Remarks
	**DelayFor**: $res \triangleq ref\ (d) \rightsquigarrow base$ When *ref* occurs (ref?), its DelayFor STA is spawned by Detect STA. The spawned STA stays in the detect location until *base* ticks *d* times. When *base* ticks *d* times (x == d), it transits to the tick location and triggers *res* (res!). Then it becomes inactive (denoted "exit()"), i.e., calculation of the current tick of *res* is completed.
	**ITE (if-then-else)**: $res \triangleq b\ ?\ c1 : c2$ ITE generates a new clock *res* that behaves either as *c1* or as *c2* base on the value of boolean variable *b*. If *b* is true (b == 1), the tick of *res* will be triggered (denoted res!) whenever *c1* occurs (c1?). Otherwise, *res* ticks with *c2* when *b* is false (b == 0).
	**FilterBy**: $res \triangleq base \blacktriangledown u(v)$ FilterBy filters the instants of *base* based on a binary word *w=u(v)*, i.e., ∀ k ∈ N , if the *kth* bit in *w* is 1, then at the *kth* tick of *base*, *res* ticks. *u* and v are two boolean arrays. *lu* and *lv* represent the size of *u* and *v*. As *base* ticks (base?), the STA firstly traverses the bits in u (at prefix state) and then iterates the bits in v (at period state). If the present bit (indicated by the index) of the binary word is 1, the STA triggers *res* (res!). Otherwise, it moves to the initial state, updates the index to refer to the next bit of *w* (i++/j++) and repeats the process.
	**Probabilistic Coincidence:** $c1 \equiv_p c2$ When *c1* (*c2*) ticks via c1? (c2?), the STA checks if the other clock, *c2* (*c1*), ticks at the same time. If c2 (c1) occurs within a positive infinitesimal value (t<=e), the STA transits to success location. Otherwise, the coincidence relation is violated and STA transits to fail location. Probabilistic coincidence is expressed as: *Pr[bound]([ ] ¬ Coincidence.fail)* $\geq p$.
	**Probabilistic Subclock:** $c1 \subseteq_p c2$ The relation limits that *c2* (superclock) must tick when *c1* (subclock) ticks, i.e., when *c1* ticks, c2 must coincide with c1. When c1 (c2) occurs, the STA checks whether the other clock also ticks at the same time. When c1 (subclock) ticks but c2 does not occur (within e time unit), the relation is violated and the STA transits to fail location. Probabilistic subclock is expressed as: *Pr[bound]([ ] ¬ Subclock.fail)* $\geq p$.
	**Probabilistic Exclusion:** $c1 \#_p c2$ When *c1* (*c2*) ticks via c1? (c2?), the STA checks if the other clock, *c2* (*c1*), ticks at the same tme, i.e., whether *c1* (*c2*) occurs or not when t ≤ e. If it occurs, the exclusion relation is violated and STA moves to fail location. Probabilistic exclusion is expressed as: *Pr[bound]([ ] ¬ Exclusion.fail)* $\geq p$.
	**Probabilistic Precedence:** $c1 \prec_p c2$ The relation states that *c1* must run faster than *c2*, i.e., the history of *c1* (*h1*) must be greater than or equal to the history of *c2* (*h2*), and *c2* must not tick when the histories of the two clocks are equal. Therefore, if *c1* ticks via c1? and *c1* runs slower (i.e., h1<h2), or c2 ticks via c2? when their histories are equal (h1==h2), the precedence relation is violated and fail location is activated. Probabilistic precedence is expressed as: *Pr[bound]([ ] ¬ Precedence.fail)* $\geq p$.

Fig. 9. STA of PrCCSL operators

tion, number of simulations) used for verification and simulation; (5) Automatic generation of probabilistic queries (introduced in Sect. 2) based on user-specified parameters; (6) Capability of performing verification and simulation on PrCCSL specifications against the integrated model and generated queries.

The GUI of ProTL is implemented by applying the Python package TKINTER [27]. The implementation of *Translator* is achieved by the ANother Tool for Language Recognition (ANTLR) [24], a parser generator that can constructs lexical parsers for a language by analyzing user-defined syntax of the language. We specified the syntax of PrCCSL in Backus-Naur Form (BNF) and apply ANTLR to generate a *parser* that can analyze and recognize encodings in the format of PrCCSL. The *parser* reads the PrCCSL specifications and generates abstract syntax trees (AST), i.e., an intermediate form that has tree structures.

By traversing AST, the information (i.e., operators and parameters) of PrCcsl can be extracted and utilized for generation of corresponding STA.

6 Experiment

To identify vulnerabilities of system to external malicious attackers, we combine the refined CAS system model (including the models of Raise protocol) with models of three different attackers. Formal verification on S/S related timing constraints (R1–R11) for the combined model is performed by Uppaal-SMC. The combined CAS model contains the stochastic behaviors in terms of the unpredictable environments (e.g., the traffic signs are randomly recognized by the leader vehicle of CAS and the probability of each sign type occurring is equally set as 16.7%), as well as the indeterministic behaviors modeled by weighted probability choices in the STA of attacks (see Fig. 7). In our setting, ls and qc are configured as 10 and 90, respectively. To estimate the probability of an attack being launched on CAS successfully, *Probability Estimation* query is applied to check the probability that the "*attack*" location in each attack STA is reachable from the system NSTA. The time bound of the verification is set as 10000. The probability of message falsification, message replaying and message spoofing attack being successfully completed by the corresponding attacker is within the range of [0.109, 0.209], [0.563, 0.663] and [0.143, 0.243], respectively.

In our experiments, S/S related timing constraints are specified in PrCcsl and transformed into STA using ProTL. Each constraint is specified as a PrCcsl *relation* (as described in Sect. 5.1) whose probability threshold is 95%. The verification results are demonstrated in Table 2, in which "√" denotes the corresponding requirement is satisfied while "×" indicates the violation of the requirement: Under the message replaying attack, all the S/S timing constraints are established as valid with 95% level of confidence. In the message falsification attack, the secrecy and integrity properties (R7 and R8), as well as three safety properties (R3–R5), are violated. The MSA damages the authenticity (R6) and secrecy (R7) of communication, and leads to the violations of four safety properties, i.e., R1 and R3–R5.

Table 2. Verification results of timing constraints under different attacks

Attacks	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	Average Time	Mem (Mb)
Message Falsification	√	√	×	×	×	√	×	×	√	√	√	40.20	57.94
Message Replaying	√	√	√	√	√	√	√	√	√	√	√	68.33	61.49
Message Spoofing	×	√	×	×	×	×	×	√	√	√	√	58.11	40.23

The experiment results indicate the severity of impacts on safety and security caused by the demonstrated attacks on CAS: No requirement is violated under MRA scenario while the MSA causes the violations of most safety properties.