Efficient and Timely Revocation of V2X Credentials

Supplemental Material

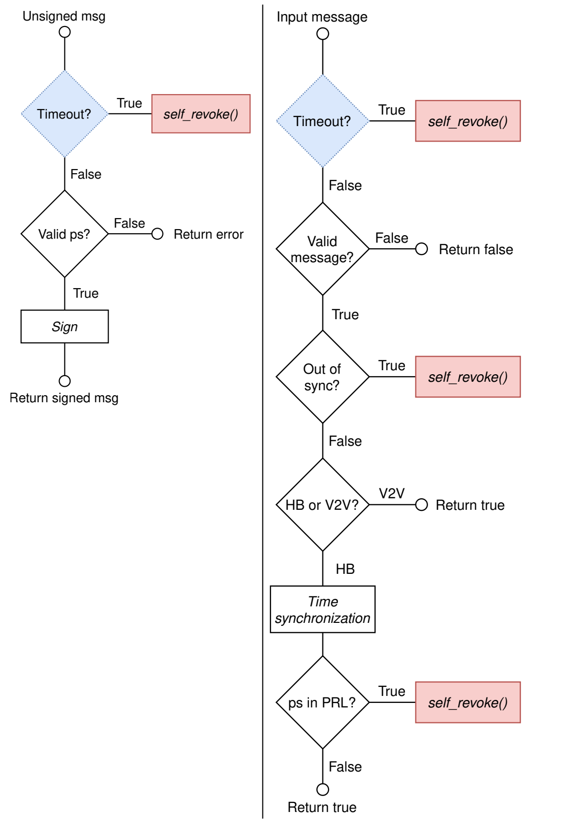
Anonymous Author(s)

A Flowcharts

Fig. 1 depicts flowcharts for the Trusted Component (TC) behavior described in our design. In particular, Fig. 1a illustrates the steps taken to sign a Vehicle-to-Vehicle (V2V) message, while Fig. 1b shows the logic to verify a network message, which can be either a heartbeat (HB) or a V2V message.

Any external message should be verified with respect to authenticity (via digital signature verification) and freshness (according to the validity window T_v). Furthermore, the TC should automatically perform self-revocation if it detects de-synchronization. For every valid HB, the TC should synchronize its internal time *now* (redundant if a local trusted time source is used), and then inspect the Pending Revocation List (PRL) to check whether self-revocation should be executed or not. For signing a message, the TC should possess valid pseudonyms in order to make a signature, which includes the input message and context metadata such as timestamps.

In Fig. 1, the *timeout* condition in blue only applies to the extension that takes into account a local trusted time source in TC, and checks if the current time is bigger than the biggest timestamp received in a HB plus T_v. Check the paper for more details.



(a) Receiving a sign request. **(b) Receiving a message.**

Figure 1: Flowcharts of the TC logic. Blue (dotted) nodes are only relevant when a local trusted time source is available in TCs.

B Tamarin Models

This appendix extends our formal verification discussion in the main paper, providing more details on how we translated our design to Tamarin and how we captured the properties of our design.

B.1 Introduction on Tamarin

Tamarin is a symbolic verification tool that supports verification of security protocols, based on multi-set rewriting rules and first-order logic. Cryptography constructs, such as symmetric/asymmetric encryption, digital signatures and hashing, are built into the tool together with a Dolev-Yao adversary.

A protocol in Tamarin is modeled as a set of *rules*, each of them defining a specific protocol step. The system's state is expressed as a multi-set of *facts*, i.e., predicates that store state information. Each rule has a pre-condition, i.e., facts that are consumed by the rule, and a post-condition, i.e., facts that are produced after its execution. In addition, *restrictions* can be defined to specify additional constraints on the rule's execution.

Rules can also produce special facts called *action facts*, which are not part of the system's state but rather are used to notify that a certain event has occurred (e.g., a HB has been generated). Action facts are used to define properties on the model, called *lemmas* in Tamarin, using first-order logic. To verify a lemma, Tamarin uses a "backwards" approach where it starts from a later state and reasons backwards to derive information about earlier states. In particular, Tamarin starts with a state that contradicts the lemma, and tries to construct

```
restriction IsLatestTime:
"
All t #i . IsLatestTime(t)@i ==>
    not (
        Ex t2 #j . TimeIncrement(t2)@j
            & j<i
            & GreaterThan(t2, t)
    )
"

rule advance_time:
[
    !Time(t)
]
--[
    /* restrictions */
    OnlyOnce(<'advance_time', t>)
    /* action facts */
    , TimeIncrement(t + '1')
]->
[
    !Time(t + '1')
]
```

Figure 2: Definition of the `advance_time` rule that increments the time counter by one, together with the `IsLatestTime` restriction that ensures the most recent value is used in a rule.

an execution trace to check whether that state is actually reachable or not via the defined rules. If a complete execution trace that reaches that state can be constructed, then the lemma is falsified, otherwise it is successfully verified.

B.2 Modeling Time

Time is modeled as a logical counter, leveraging the *multiset* built-in. To increment the counter, we defined a rule that takes as pre-condition the current value and increments it by one (Fig. 2). We also allow TC and Revocation Authority (RA) to perform an arbitrary number of operations within the same time step.

To model such counters, we use *persistent facts*, i.e., facts that remain in memory persistently, even after being consumed by a rule. In our experience, this was a better choice than using *linear facts* which, once consumed, are deleted from the system's state; Persistent facts allow reusing the same `!Time(t)` fact in multiple rules, which reduces the number of possible states and thus the memory used by Tamarin. We also experienced some undesired side-effects of using linear facts, such as infinite recursion due to the fact that rules were both consuming and producing `Time(t)` facts at the same time.

However, the use of persistent facts cause multiple `!Time` facts to be in the system's state at the same time. For example, after the rule `advance_time` in Fig. 2 is executed for the first time, we would have both `!Time('1')` and `!Time('1'+'1')` facts in memory. This would potentially cause execution of rules using the old `!Time` fact instead of the new one. Although this inconvenience does not compromise the overall accuracy of the model, it does not represent a realistic scenario and it also causes a higher number of possible states. Therefore, we added a restriction `IsLatestTime(t)` (defined in Fig. 2) in every rule to ensure that the latest time is always used.

B.3 Modeling the PRL

Similarly to time and epochs, the PRL was modeled as persistent facts as well, using restrictions to enforce that the most recent facts are used. Concerning the PRL, other than the list itself we used sequence numbers to check which fact was most recently generated. In the `Init` rule, which is enforced to be executed before any of the TC and RA rules, the PRL is initialized with a multiset containing a dummy value, since we cannot model empty multisets. Then, the rule `RA_issue_revocation` adds a pseudonym to the list, creating a new `!PRL` fact.

B.4 Modeling T_v

We modeled T_v as a parameter initialized in the `Init` rule. Our goal was to model T_v as an arbitrary value to prove our properties for any value greater than zero. To do so, we defined a first rule called `init_parameters`, which creates a linear fact `TvTmp` with initial value `'1'`. Then, we defined anoter rule called `increment_Tv` that consume such linear fact and produce a new one with value incremented by one. Finally, the `Init` rule takes the current value and finalizes it, creating a persistent fact `!Parameters` used in other rules.

Fig. 3 shows the flow described above. Since the execution of a rule is non-deterministic, the `increment_Tv` rule can be called zero to infinite times before the `Init` rule, thus giving T_v an arbitrary value between $[1 : \infty)$

B.5 Modeling Cryptographic Operations

We relied on the *signing* built-in to generate and verify signatures of HBs and V2V messages. In the `Init` rule, we generate a keypair for the RA and one for the TC's pseudonym. Private keys are kept in the system's state with the assumption that attackers have no access to them, while public keys are made available to attackers by sending them to the `Out` channel.

B.6 Mapping our Design to Tamarin

To map our design to Tamarin, we used only five main rules:

- `RA_generate_heartbeat`: This rule takes as input the current time and PRL and produces a new HB as output, signed with the RA's private key. The HB is also sent out to the network channel, and an action fact

```
// set initial value to 1
 rule init_parameters:
[
]
--[
    /* restrictions */
    OnlyOnce('init_parameters')
]->
[
    TvTmp('1')
]

// non-deterministic increment
rule increment_Tv:
[
    TvTmp(tv)
]
--[
]->
[
    TvTmp(tv + '1')
]

// finalize value
rule Init:
[
    TvTmp(tv)
]
--[
    /* restrictions */
    OnlyOnce('Init')
    /* action facts */
    , SystemInitialized(tv)
]->
[
    !Parameters(tv)
]
```

Figure 3: Model of parameter T_v with arbitrary value.

HeartbeatGenerated(HB) is produced. This rule can only be executed once per PRL and time step to reduce the number of states, since multiple execution with the same parameters would generate the same HB, which is not relevant in our model since Tamarin already allows replays of network messages.

- RA_issue_revocation: Taking as input the current time, PRL, and the pseudonym's public key, this rule generates a new PRL by adding the pseudonym to the list and incrementing the sequence number by one. We added a restriction to revoke each pseudonym only once. After execution, the action fact RevocationIssued(ps, t) is generated.
- TC_process_heartbeat: The TC processes a HB taken as input from the channel. Aside from the usual checks on time, restrictions ensure that the received HB has a valid signature and timestamp. In addition, the TC has to check whether the pseudonym is in the PRL or not. To do so, we divided this rule in two mutually-exclusive rules that produce different results: if the pseudonym is in the PRL, the action fact Revoked(t) is produced, otherwise it is not. Either way, a HeartbeatProcessed(HB) action fact are produced, along with a new !Time(t_hb) persistent fact to synchronize the TC time.
- TC_sign_message: The TC generates a new V2V message signed with the pseudonym's private key. It takes the latest time as input, while restrictions ensure that the TC has not been previously revoked (i.e., the Revoked(t) action fact does not exist). The rule produces a signed timestamped message and sends it to the out channel. A Signed(msg,ps) action fact is produced as well.
- process_message: This rule models a generic third entity that receives a V2V message from network and verifies it. If the signature is valid and the message is fresh, a MessageAccepted(msg,ps,t) action fact is generated.

```
// property (I)
lemma effective_revocation [heuristic=o "oracle.py"]:
"
All msg ps t #i . MessageAccepted(msg, ps, t)@i ==>
    Ex tv #j . SystemInitialized(tv)@j & j<i
    &
    not (
        Ex t_rev #k . RevocationIssued(ps, t_rev)@k
        & GreaterThan(t, t_rev + tv)
    )
"

// property (II)
lemma no_heartbeats_processed_after_tolerance [heuristic=o "oracle.
     py"]:
"
// for all heartbeats processed
All prl t_hb #i . HeartbeatProcessed(<prl, t_hb>)@i ==>
    Ex tv #j . SystemInitialized(tv)@j & j<i
    &
    // t_hb is <= t_rev + tv
    not (
        Ex ps t_rev #k . RevocationIssued(ps, t_rev)@k
        & k<i
        & GreaterThan(t_hb, t_rev + tv)
    )
"
```

Figure 4: Properties our main design translated to Tamarin lemmas.

B.7 Lemmas

Aside from sanity and functional lemmas, which have the purpose to ensure the correctness of the model, the lemmas in Fig. 4 show our properties defined in the paper translated to our Tamarin models.

The effective_revocation lemma ensures that a third-party entity (e.g., another TC) stops accepting all messages from a revoked TC as soon as its internal time passes $t_{rev} + T_v$. This proves that the revoked TC cannot update its internal time *now* to a value greater than t_{rev}, meaning that it will eventually get de-synchronized after the validity window T_v has passed. However, since TCs in the network may update their internal time *now* at different times (depending on the reception of HBs), the effective revocation time is not absolute, because each TC may take a different amount of time to pass $t_{\text{rev}} + T_v$. Nevertheless, since in the paper we assume that honest TCs can be *at most* T_v behind the RA time, the translation from this lemma to the effective revocation time T_{eff} automatically follows, i.e., $T_{\text{eff}} = 2T_v$.

The lemma `no_heartbeats_processed_after_tolerance`, instead, proves that a revoked TC cannot process *any* HBs whose timestamp is bigger than $t_{\text{rev}} + T_v$. There are two possible reasons why this happens: *(1)* The TC has processed a HB generated since t_{rev}, meaning that it is in a revoked state since then; *(2)* The TC has not processed any HB generated since t_{rev}, meaning that processing a HB with timestamp bigger than $t_{\text{rev}} + T_v$ would trigger automatic revocation. Either way, this can be directly translated to our second property: Pseudonyms belonging to the revoked TC can be safely removed from the PRL after $t_{\text{rev}} + T_v$, i.e., $T_{\text{prl}} = T_v$.

In order to build an efficient proof for some of our lemmas, we gave as input to Tamarin a Python script called *oracle*, which provides a custom ranking for *goals*. Indeed, Tamarin proves a lemma by attempting to solve the open goals of the constraint system, which are sorted based on some heuristics. By providing an oracle as input, goals can be ranked according to a custom heuristics. This was essential in our case, because for some of the lemmas Tamarin could not terminate with the default heuristics due to an inefficient ranking of goals. For example, solving `!Time` goals needs to be postponed as much as possible because, in an unconstrained system, time can have an arbitrary value and Tamarin would get stuck in an infinite loop. By using a specific ranking, instead, we could give priority to other goals that add more and more constraints, allowing `!Time` goals to be solved efficiently. Thanks to our oracles, the lemmas in our model are proven in only a few minutes.

B.8 Trusted Time Source in TCs

The design that models a trusted time source in TC comes with a major difference, i.e., that TC and RA use the same `!Time` facts. That is, this means that their clock is perfectly synchronized, and the TC does not need to process HBs to advance its internal time. This change caused significant differences in our lemmas, as shown in Fig. 5.

First of all, the `effective_revocation` lemma can now directly prove our first property to calculate the exact value for T_{eff}. Indeed, we now assume that all TCs are perfectly synchronized with the RA time, hence the revocation time is now absolute and comes at $t_{\text{rev}} + 2T_v$.

```
// property (I)
lemma effective_revocation [heuristic=o "oracle.py"]:
"
All msg ps t #i . MessageAccepted(msg, ps, t)@i ==>
    Ex tv #j . SystemInitialized(tv)@j & j<i
    &
    not (
        Ex t_rev #k . RevocationIssued(ps, t_rev)@k
        & k<i
        & GreaterThan(t, t_rev + tv + tv)
    )
"

// property (II)
lemma no_operations_after_timeout [heuristic=o "oracle.py"]:
"
// for all operations done by the TC
All t #i . NewOperation(t)@i ==>
    Ex tv #j . SystemInitialized(tv)@j & j<i
    &
    // this operation was done not after t_rev + tv
    not (
        Ex ps t_rev #k . RevocationIssued(ps, t_rev)@k
        & k<i
        & GreaterThan(t, t_rev + tv)
    )
"
```

Figure 5: Properties of our design with a trusted time source in TC translated to Tamarin lemmas.

Scenario	Time?	T_v (s)	T_{eff} (s)	T_{prl} (s)
A1	no	30	60	30
A2	no	150	300	150
B1	yes	30	60	30
B2	yes	150	300	150

Table 1: Scenarios evaluated in our simulation. Parameters are chosen according to the desired T_{eff}. The column named *Time?* indicates whether a local trusted time source is used in TCs or not.

Secondly, thanks to internal timeouts in TC, we can have a stronger proof for the second property, saying that not only the TC does not process HBs generated after $t_{\text{rev}} + T_v$, but also that the TC cannot perform *any* operations after that time because the automatic revocation timeout would be triggered. To prove this, we defined an additional rule called `TC_do_operation`, which models a generic operation performed by the TC via the `NewOperation(t)` action fact. This is then used in `no_operations_after_timeout` to prove that there cannot be `NewOperation(t)` action facts with a value for t bigger than $t_{\text{rev}} + T_v$.

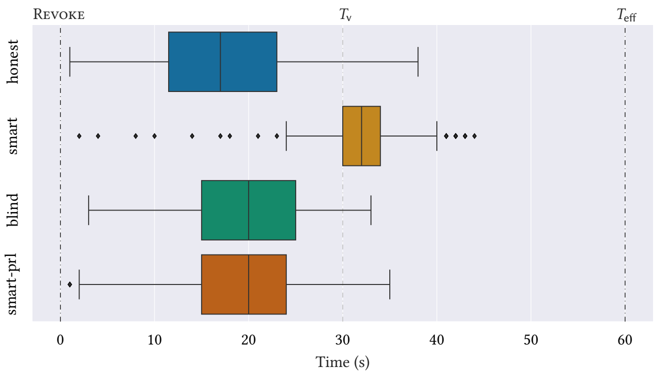


Figure 6: Simulation results for scenario A1.

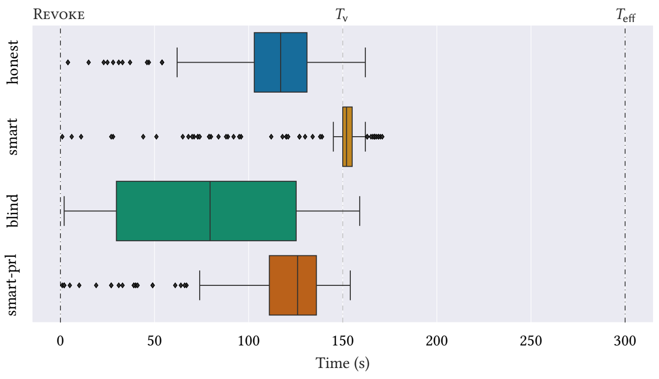


Figure 7: Simulation results for scenario A2.

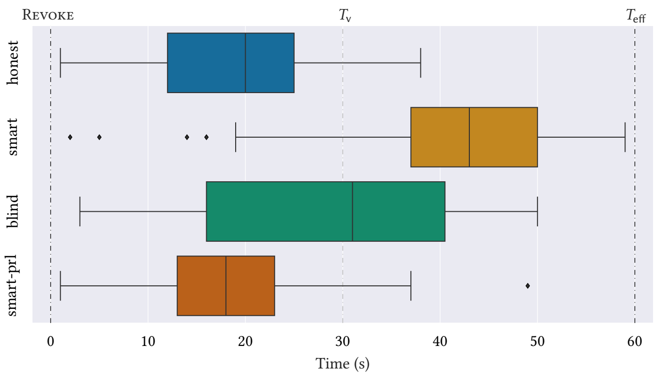


Figure 8: Simulation results for scenario B1.

C Complete Simulation Results

We ran simulations for each of the scenarios described in Tab. 1. Results are depicted in Figs. 6 to 9.

Recall from the paper that each value of the boxes represents the time between the revocation of *ps* (REVOKE event) and the last V2V message signed with *ps* that was verified

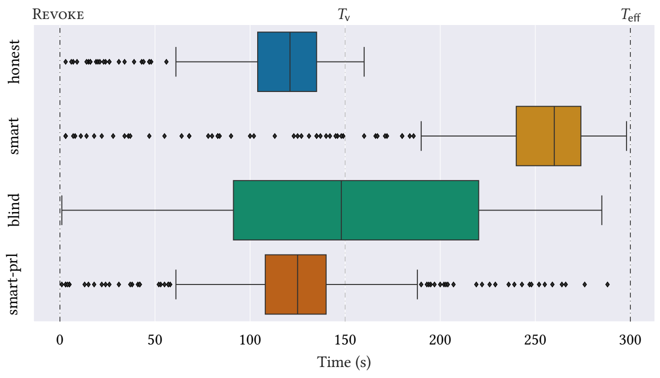


Figure 9: Simulation results for scenario B2.

by a non-malicious TC (VERIFY event). The latter time represents the *effective revocation time* for that particular pseudonym *ps*. The box plots give the distribution of such values, obtained aggregating more than 600 revocations, filtering out negative values.

The figures show one significant difference between the main design (A1 and A2) and the extension with a local trusted time source (B1 and B2): While in the former most revocations are effective around or before T_v, in the latter attackers are able to postpone revocation up until T_{eff} in some cases, i.e., it appears easier to reach the upper bound, especially for powerful attackers such as the *smart* one. This, is due to the fact that, while in the main design revoked TCs cannot synchronize their time since t_{rev}, with a local time source TCs can still advance their internal time up until $t_{rev} + T_v$, before triggering the automatic revocation logic. That is, in the latter case a revoked TC is able to generate "more fresh" V2V messages, which can be processed by other TCs later in time.

This peculiarity suggests that a local trusted time source negatively affects the revocation time. While this may be true in the average case, it still does not affect the worst-case effective revocation time T_{eff}, as shown in the figures.

D Calculating Expected Size of the PRL

This appendix covers the basics of the utilized probabilities to then explain the Markov matrix. We also describe the calculation of the stationary distribution as a means to calculate the expected PRL size. Finally, we discuss the computation of the probabilities and expected values used in the evaluation.

D.1 Utilized Probabilities

Recall from the main paper that we model the processes of gaining and losing pseudonyms from the PRL as two individual probabilities and combine them in the next step into a Markov chain, according to the following equations:

$$G_{i,k} = \binom{n-i}{k} \cdot p^k \cdot (1-p)^{n-i-k} \tag{1}$$

$$L_{i,k} = \binom{i}{k} \cdot (\frac{1}{T_{\text{prl}}})^k \cdot (1 - \frac{1}{T_{\text{prl}}})^{i-k} \tag{2}$$

$$p_{i,j} = \sum_{l=Max(i-j,0)}^{i} L_{i,l} \cdot G_{i,l-i+j} \tag{3}$$

The two probabilities shown in Eq. (1) and Eq. (2) are denoted as G_{il} and L_{il} respectively and can be seen as probability matrices that at location il have the probability for being in state i and gaining or losing l pseudonyms from the list. The core assumption for these two probabilities is that both adding and removing pseudonyms from the revocation list can be seen as a binomial distribution that only depends on: *(1)* the current size of the list, i.e., how many certificates there are left to be revoked or removed from the list, *(2)* the probability for each pseudonym to be revoked at any given time, and *(3)* the time that a pseudonym stays in the list. As such, this model assumes that pseudonyms are renewed as soon as they are evicted from the PRL, i.e., the list can never hold more than the total number of n pseudonyms, which is a realistic assumption for large n as it is unlikely that so many pseudonyms in the system would be revoked in a short time during regular operation. Furthermore, in this way we model the process of losing pseudonyms from the list as a probabilistic process that has the expected value at T_{prl}. We, however, can not model a precise eviction of the pseudonym from the PRL after T_{prl} time steps. In our view this is an acceptable tradeoff as on average and over the lifetime of the PRL, the modeled behavior comes very close to the actual behavior of evicting pseudonyms from the PRL after exactly T_{prl} time steps.

D.2 Markov Model

We can now combine these two probabilities into a matrix that at location p_{ij} has the probability of moving from state i to state j. Starting with the simple example of $n = 3$ vehicles, multiple entries of this matrix are straightforward:

- Starting at any state i and stepping into state $j = 0$ requires to not revoke any new pseudonyms and to lose all existing pseudonyms from the list. This is the combination of the two probabilities $L_{ii}G_{i0}$. For example, going from state $i = 2$ to $j = 0$ requires to lose 2 pseudonyms when there are 2 in the PRL, and requires to gain no new pseudonyms when there are already 2, leading to the two probabilities $L_{22}G_{20}$
- Starting at state $i = 0$ and stepping into a state j implies that no pseudonyms can be lost from the PRL and j pseudonyms have to be added.

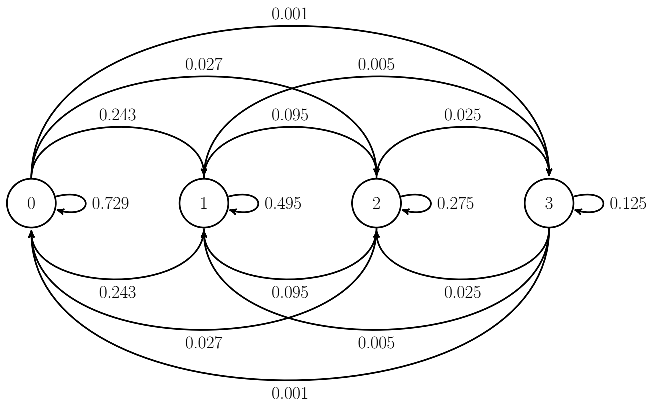


Figure 10: A graph of the Markov chain with the exemplary parameters of $n = 3$, $p = 0.1$, and $T_{prl} = 2$.

- Similarly, starting in state $i = 3$ and stepping into any state j implies that no pseudonyms can be gained, but $j - i$ pseudonyms are removed from the PRL.

The most interesting state transitions in this probability matrix are the transitions that consist of multiple combinations of events. One example is the transition from $i = 2$ to $j = 1$. Here, we first have the obvious possibility that we reach the state $j = 1$, i.e., one pseudonym in the PRL, by losing one pseudonym and gaining none. However, we also have the possibility to lose both pseudonyms in the PRL and gain one, leading to still one remaining pseudonym in the PRL. This is because the pseudonyms in the PRL and outside of the PRL are independent and pseudonyms can still be revoked while others are removed from the list. Thus, the state transition consists of the following parts: $p_{21} = L_{21}G_{20} + L_{22}G_{21}$.

The full state probability matrix for $n = 3$ vehicles can be seen below. Note that a matrix for n vehicles is a $n + 1 \times n + 1$ matrix to accommodate the state of the empty PRL.

$$P = \begin{bmatrix} L_{00}G_{00} & L_{00}G_{01} & L_{00}G_{02} & L_{00}G_{03} \\ L_{11}G_{10} & L_{10}G_{10}+L_{11}G_{11} & L_{10}G_{11}+L_{11}G_{12} & L_{10}G_{12} \\ L_{22}G_{20} & L_{21}G_{20}+L_{22}G_{21} & L_{20}G_{20}+L_{21}G_{21} & L_{20}G_{21} \\ L_{33}G_{30} & L_{32}G_{30} & L_{31}G_{30} & L_{30}G_{30} \end{bmatrix}$$

Such probability matrices are also called discrete time Markov chains [1]. Fig. 10 depicts the transition graph for the above Markov chain when we assume the parameters of $p = 0.1$ and $T_{prl} = 2$. Based on this first small example, we can approach a closed formula for an arbitrary number of pseudonyms (n). This closed formula is shown in Equation (3). The core idea is that each entry depends on a combination of gaining and losing pseudonyms based on the maximum of $i - j$ and 0 and ranges up to i. Then, any state combination that is not possible will be 0 through the binomial coefficient. However, this range is necessary to catch the complicated combinations in the center of the matrix. Each combination is

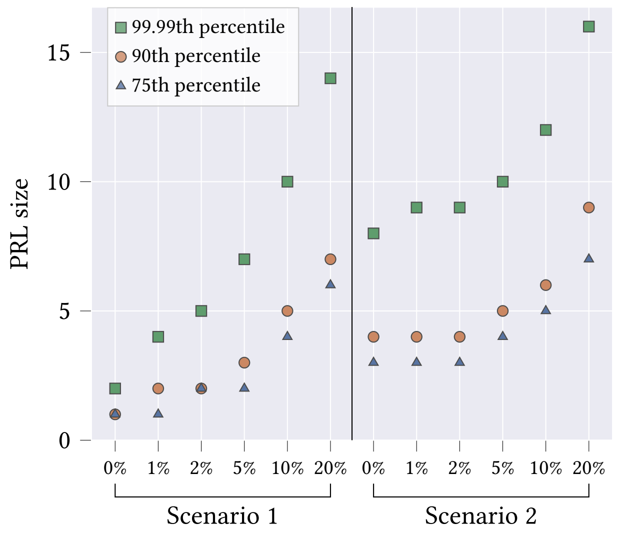


Figure 11: Markov model percentiles for maximum PRL sizes for $T_{prl} = 30s$, $n = 800$, and different shares of attackers in each baseline revocation scenario.

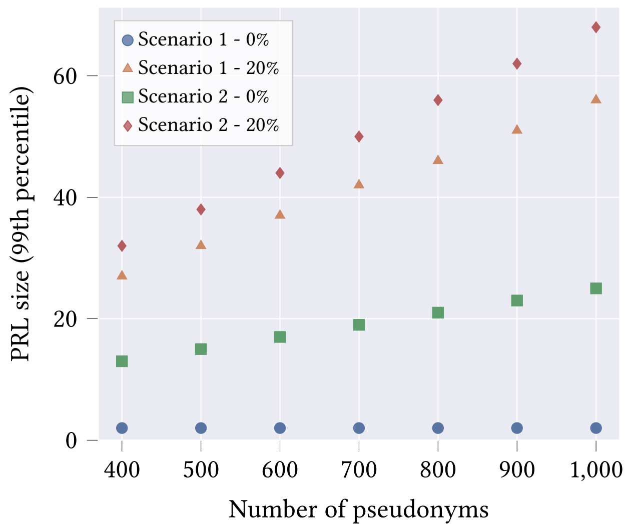


Figure 12: 99th percentile PRL sizes for fixed $T_{prl} = 270$ and varying number of pseudonyms (n) under four scenarios.

then based on losing l pseudonyms from the list and gaining $l - i + j$ pseudonyms, which basically iterates through the combinations with l as a stepping variable.

D.3 Calculating the Expected PRL Size

In discrete time Markov chains, a probability state vector π can be multiplied with the Markov matrix P to gain the probabilities for π after a single transition in the model [1] as follows: $\pi' = \pi \cdot P$. This, however, is only sufficient to gain the state probabilities after specific amounts of steps. In contrast to this approach, Markov chains may approach a so-called state equilibrium which is a stationary probability distribution that does not change even after taking a step in the Markov chain. The stationary distribution follows the equation $\pi = \pi \cdot P$ [1].

To gain this stationary distribution, we solve the following equation system:

$$\begin{aligned} &\pi = \pi P \\ &\leftrightarrow \pi - \pi P = 0 \\ &\leftrightarrow \pi(I - P) = 0 \\ &\leftrightarrow (I - P)^T \pi^T = 0 \end{aligned}$$

Lastly, since there may be several solutions to this linear equation, we add the specific constraint to this equation system that the sum of π is equal to 1. This is the case for each probability vector in Markov models [1], and follows the intuition that, from any state probability, the probability to reach any other state is 1. To add this constraint, we add a row of 1 to $(I - P)^T$ and append a 1 to the zero vector.

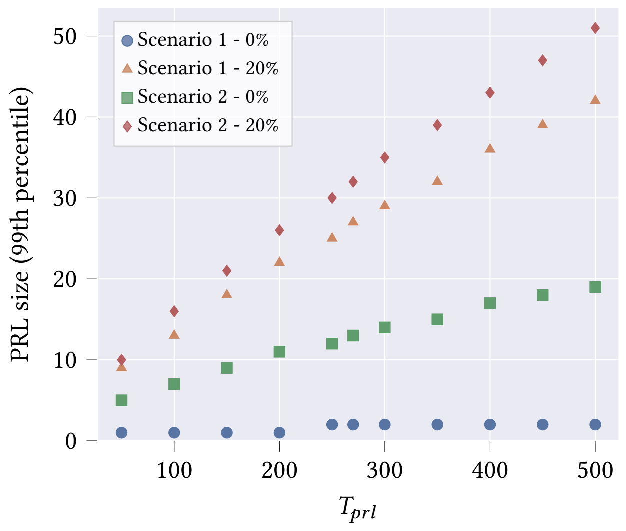


Figure 13: 99th percentile PRL sizes for fixed $n = 400$ pseudonyms and varying T_{prl} under four scenarios.

Solving this linear equation yields a stationary distribution that is stable. Fig. 11 summarizes this stationary distribution by showing the list sizes that occur to 75%, 90%, and to 99.99%. The graph shows these percentiles under different scenarios, i.e., with different shares of attackers in the network.

In the following, we expand this evaluation with Figures 12 and 13. Fig. 12 depicts only the 99th percentile for a fixed T_{prl} and only focuses on the four extreme cases: for each of the two baseline scenarios, 0% and 20% of attackers in the network. The horizontal axis then depicts a growing number of pseudonyms (and, therefore, vehicles) that participate in the network. This is to show that a larger number of vehicles does not exponentially grow the expected size of the PRL. Instead, the 99th percentile of the list size grows linearly with n.

Similarly, Fig. 13 shows the same situation for a fixed number of pseudonyms at $n = 400$ but a varying window for T_{prl}. This second graph shows that the PRL size also grows linearly with a linear increase in the time that a pseudonym stays in the PRL.

D.4 Probabilities and Expected Revocations

Above, p is the probability of revocation in each time step. As time steps are at the granularity of seconds, and our system runs for a long time, it turns out that the per-step probabilities leading to realistic revocation rates are very small. Instead, the probabilities used in the PRL size evaluation were described in terms of a compound probability q that a revocation occurs at least once over a number of s steps. This probability can be computed via the geometric series

$$q(p, s) = \sum_{i=1}^{s} p \cdot (1-p)^{i-1} = 1 - (1-p)^s$$

which sums up the probabilities that the first revocation occurs the first time in step $i \in [1 : s]$. Note that this is equal to 1 minus the probability that in all s steps no revocation occurs. Solving for p we obtain

$$p(q, s) = 1 - \sqrt[s]{1-q}\,.$$

We used this formula to compute the baseline per-step probabilities underlying our scenarios for honest vehicle and attacker separately. The final probabilities p for the Markov models were then obtained by averaging the probabilities weighted according to the share of attackers.

For estimating the expected numbers of revocations in our scenarios we needed to calculate the expected value for n pseudonyms over s steps. If each step was independent this would simply be $n \cdot s \cdot p$, however, as explained above, in our Markov model at most n pseudonyms can be revoked at a time. We compensated for this fact by taking into account the chance p_{prl} that a given pseudonym is on the PRL at any given time and computed

$$E_{\text{rev}} = n \cdot s \cdot p \cdot (1 - p_{\text{prl}})$$

as the desired estimate. We set p_{prl} to the median PRL size, as obtained by the Markov model for each scenario, divided by n.

References

[1] Holger Hermanns. 2002. Markov Chains. In *Interactive Markov Chains*. Springer, 35–55.