Statistical Model Checking of Guessing and Timing Attacks on Distance-bounding Protocols

Musab A. Alturki*, Max Kanovich†‡, Tajana Ban Kirigin§, Vivek Nigam¶††, Andre Scedrov‖‡ and Carolyn Talcott**
* King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia
† University College London, London, UK
‡ National Research University Higher School of Economics, Moscow, Russia
§University of Rijeka, Department of Mathematics, Croatia
¶Federal University of Paraíba, João Pessoa, Brazil
‖University of Pennsylvania, Philadelphia, PA, USA
**SRI International, Menlo Park, CA, USA
††fortiss, Munich, Germany

***Abstract*—Distance-bounding (DB) protocols were proposed to thwart relay attacks on proximity-based access control systems. In a DB protocol, the verifier computes an upper bound on the distance to the prover by measuring the time needed for a signal to travel to the prover and back. DB protocols are, however, vulnerable to distance fraud, in which a dishonest prover is able to manipulate the distance bound computed by an honest verifier. Despite their conceptual simplicity, devising a formal characterization of DB protocols and distance fraud attacks that is amenable to automated formal analysis is non-trivial, primarily because of their real-time and probabilistic nature. In this work, we present a framework, based on rewriting logic, for formally analyzing different forms of distance-fraud, including recently identified timing attacks. We introduce a generic, real-time and probabilistic model of DB protocols and use it to (mechanically) verify false-acceptance and false-rejection probabilities under various settings and attacker models through statistical model checking with MAUDE and PVESTA. Using this framework, we first accurately confirm known results and then define and quantitatively evaluate new guessing-ahead attack strategies that would otherwise be difficult to analyze manually.**

Keywords: Distance-bounding protocols, Distance fraud, Probabilistic rewriting, Statistical model checking, MAUDE

I. INTRODUCTION

Proximity-based access is a typical security requirement in many real-world cyber-physical systems, e.g., smart-card gate-access control systems. Proximity-based access, however, is well-known to be vulnerable to *relay* (or *Mafia fraud*) attacks, in which an attacker relays messages between a verifier (e.g., a card reader) and a prover (e.g., an NFC tag), maliciously extending the effective range of communication. To protect against relay attacks, distance-bounding (DB) protocols have been introduced [1], [2]. In a DB protocol, the verifier computes an upper bound on the distance to the prover by measuring the time needed for a signal to travel to the prover and back. Since a signal's velocity cannot exceed the speed of light, an upper bound on the distance to the prover can be computed. The round-trip time is measured through a rapid bit exchange, in which the verifier sends a challenge bit to the prover and the prover responds as quickly as possible with the corresponding response bit. Since relaying challenges and responses during this phase introduces relatively significant delays, an attacker's ability to mount a relay attack is severely limited.

DB protocols are, however, susceptible to Distance Fraud, in which a dishonest prover is able to manipulate the distance bound computed by an honest verifier to make himself appear closer than he actually is, causing the verifier to falsely accept the prover. Distance fraud attacks are *timing* attacks that are mounted without colluding with any external entity, which renders them particularly dangerous. One category of attacks that can lead to distance fraud is *guessing* attacks, in which the attacker attempts to correctly guess the response to the verifier's challenge [1]. Guessing can potentially be used to send out a response in advance (before the challenge arrives) so that the measured round-trip time is reduced. A second category of distance fraud that has been recently identified is the *in-between-ticks* attack [3], which exploits the computational limitations of a verifier and the differences between the physical time and the discrete time of the verifier's clock. Both attack categories build on how sensitive DB protocols are to timing of events, since the slightest timing manipulations can result in significant errors in computing the distance. It is important to note that both categories of attacks apply to the rapid bit exchange phase of almost all DB protocols, including those of the Brands and Chaum [1] and the Hancke and Kuhn [2] families of protocols.

Although guessing attacks, in particular, were identified when the first DB protocol was proposed [1], little has been done since then to investigate formally and systematically their different strategies and countermeasures. Appropriately timing guessed responses and using different guessing strategies can significantly affect the chances of successfully mounting a guessing attack. Moreover, as is typical of manual analysis, the analytical models developed by hand of the recently identified in-between-ticks attacks [3] had to rely on some simplifying assumptions to make the analysis tractable. Furthermore, these models are necessarily high-level, abstracting away many

operational details of the protocol and the threat model that could potentially affect the probability of an attack. Manually investigating various attack strategies while taking into account relevant details can be too labor-intensive and is susceptible to human error.

Despite their conceptual simplicity, devising a formal characterization of DB protocols and distance fraud attacks that is amenable to automated formal analysis is non-trivial, primarily because of their real-time and probabilistic nature. To systematically and formally analyze timing attacks on DB protocols, an expressive logical formalism is needed in which *general and faithful* models of these protocols capturing a variety of possible behaviors can be developed. Furthermore, the models need to be both *timed* and *probabilistic*, so that randomized behaviors, environment uncertainties (such as noise) and timing of events can formally be described and analyzed. Additionally, to automate this analysis, the models must be *executable*, enabling quick prototyping and experimentation with different designs and configurations. Executability is particularly important as DB protocols are notorious for being hard to implement in practice. Finally, the formalism in which these models are developed needs to support efficient formal analysis of *quantitative properties*, including probabilities, to reason about resilience against distance fraud.

In this work, we present a framework that satisfies these requirements for formally analyzing different forms of distance-fraud attacks on DB protocols and their countermeasures. The framework is based on probabilisitic rewrite theories [4] enabling a formal yet natural representation of DB protocols and attacker behaviors. Using MAUDE [5] and PVeStA [6], statistical model checking of quantitative properties of a DB protocol model can be automatically performed. This methodology provides an efficient and automatic means of verifying complex systems without having to make simplifying assumptions that are usually needed for full probabilistic analysis to be tractable. Furthermore, the framework is quite versatile, allowing the estimation of false-acceptance and false-rejection probabilities under various settings and attacker models. It provides an effective means for quantitatively evaluating the resilience of different DB protocol designs against various forms of distance fraud, a process that would be too difficult to handle manually. We first show how the framework is used to accurately and mechanically confirm known results about simple guessing and in-between-ticks attacks. We then define new guessing-ahead attack strategies that have not been thoroughly investigated before, and then evaluate them with and without noise. The analysis enabled by our framework provides deeper insights on how attack strategies compare against each other in realistic settings and how these timing attacks are best thwarted. Moreover, it generally demonstrates how quantitative evaluation based on formal models of DB protocols can be immensely useful.

The rest of the paper is organized as follows. In Section II, we describe the Hancke-Kuhn protocol, as a representative DB protocol, along with distance fraud attacks. Section III introduces in some detail our rewriting model of DB protocols. Then, in Section IV, we define the properties to be analyzed and discuss the statistical model checking analysis results. This is followed by an overview of related work in Section V. The paper concludes with a summary and a discussion of future work in Section VI.

II. DISTANCE-BOUNDING PROTOCOLS

The goal of a distance-bounding protocol is to ensure access to some resource to valid provers that are within a specified distance bound, and, at the same time, reject access to provers that are located outside of the distance bound perimeter. There are typically three phases of a DB protocol, an initialization or a setup phase, during which nonces and/or commitments are calculated and/or exchanged, followed by a distance measurement phase that establishes the physical distance, and a finalizing phase used to check commitments, i.e., confirm identification. The distance measurement phase is the essential part of any DB protocol. Both guessing and in-between-ticks attacks, as the main focus of our analysis, are due to failures in the distance measurement phase.

A. The Hancke-Kuhn Protocol

As a DB protocol, the Hancke-Kuhn protocol [2] aims to ensure that the prover, P, is in the vicinity of the verifier, V. This protocol assumes that the prover and the verifier share a long term secret key, K, and a public hash function, h.

In the initial (setup) phase of the protocol the verifier and the prover generate nonces N_V and N_P which are used to calculate a sequence of $2n$ bits using K and h: $R_1^0, \ldots, R_n^0 || R_1^1, \ldots, R_n^1,\ R_i^j \in \{0,1\}$.

The setup phase of Hancke-Kuhn protocol is followed by a series of n single bit exchanges, defined by the following procedure: To a random challenge bit C_i sent by the verifier in the ith round, the prover instantly replies with either R_i^0, in case $C_i = 0$, or R_i^1, in case $C_i = 1$. At the same time, the prover discards the corresponding other bit, R_i^1 or R_i^0. This way, the prover reveals only half of all the bits derived in the initial phase. For each round, the verifier marks the time when a challenge bit is sent, and the time the response is received.

In the last phase of the protocol, the verifier computes his distance from the prover and checks that the responses are correct. The verifier grants access to the prover if all time tests for bit exchanges are successful, i.e., do not exceed the predefined distance bound, and if all n bits are correctly exchanged. Keeping in mind potential errors, due to e.g., noise, the verifier's decision can be parametrised so that the access is granted if the time-test is satisfied in a number of rounds, e.g., in a simple majority of rounds, and if only a number of response bits, k out of n, are correct. Different acceptance criteria are further explored when we describe the rewriting model of DB protocols in Section III.

B. Distance Fraud Attacks

Guessing Attacks. The responder involved in the distance measurement phase of a DB protocol can be an *attacker*, who does not share the relevant secrets, including the secret key K,

with the verifier, as well as a dishonest prover, having access to the shared secrets. An attacker with no access to the shared secrets may try to successfully complete a protocol session by *randomly* guessing the correct bit responses to the challenge bits received. A dishonest prover may also use guessing to try to appear closer than he actually is. He does so by guessing challenge bits and sending response bits ahead of time, before receiving the challenge bits. This affects the measured time difference, and hence the relative distance calculated by the verifier. Furthermore, as he knows the $2n$ bit sequence used in the protocol session, the dishonest prover may perform *educated guessing*: before receiving a challenge bit C_i, he may randomly choose between potential response bits R_i^0 and R_i^1. Moreover, in the case when $R_i^0 = R_i^1$, the correct response is a priori known to the prover.

In-Between-Ticks Attack. Another type of attack that exploits the distance measurement phase of a DB protocol is the *In-Between-Ticks attack* identified in [3]. It does not involve guessing of the response bits, and can appear even when the prover is honest and adversary is not present. This attack is a consequence of the foundational difference between real-time in the physical world and time management by discrete time processors that are used as verifiers. Namely, such a discrete time verifier performs instructions and measures time following his clock-cycle rate and performance limitations. This can result in discrepancy between the actual and the observable time intervals, i.e., between actual time when the bits are sent and received, on one side, and the recorded time of sending and receiving bits, on the other. For more details on this type of attack, including its probability analysis, see [3].

III. DB Protocols as Timed Probabilisitic Rewrite Theories

Formal verification of resilience of a DB protocol against guessing and timing attacks requires having an expressive and executable formal model of the protocol and the attacker behaviors. Rewriting logic [7], and its probabilistic extensions [8], [4], provide a capable underlying formalism for this purpose. Furthermore, using Maude [5], a high-performance rewriting logic engine, models can be simulated to generate random sample executions that can be used to statistically model-check probabilistic properties of the protocol.

A. Rewrite Theories and Maude

A *rewrite theory* $\mathcal{R}$ formally describes a concurrent system including its static structure and dynamic behavior. It is a tuple $(\Sigma, E \cup A, R)$ consisting of: (1) a membership equational logic (MEL) [9] signature Σ that declares the kinds, sorts and operators to be used in the specification; (2) a set E of Σ-sentences, which are universally quantified Horn clauses with atoms that are either equations $(t = t')$ or memberships $(t : s)$; (3) a set A of equational axioms, such as commutativity, associativity and/or identity axioms; and (4) a set R of rewrite rules $t \longrightarrow t'$ if C specifying the computational behavior of the system. (See [10] for a detailed account of generalized rewrite theories).

Probabilistic rewrite theories extend regular rewrite theories with probabilistic rules [8], [4]. Assuming $\overrightarrow{x}$ and $\overrightarrow{y}$ are disjoint, a probabilistic rewrite rule has the following form:

$$(\forall \overrightarrow{x}, \overrightarrow{y})\ r : t(\overrightarrow{x}) \longrightarrow t'(\overrightarrow{x}, \overrightarrow{y}) \text{ if } C(\overrightarrow{x}) \quad \text{with probability } \overrightarrow{y} := \pi(\overrightarrow{x})$$

A probabilistic rule introduces on its right-hand-side term new variables $\overrightarrow{y}$, the values of which depend on a probability distribution function π parametrized by $\theta(\overrightarrow{x})$, where θ is a matching substitution satisfying the condition C. A canonical example is the probabilistic rule specifying a battery-operated clock [5]:

$$\begin{aligned} clock(t, c) \longrightarrow\ & \textbf{if } B \textbf{ then } clock(t + 1, c - c/1000.0) \\ & \textbf{else } broken \\ & \text{with probability } B := Bernoulli(c/1000.0) \end{aligned}$$

As the clock $clock(t, c)$ ticks (t is the current time and c is the remaining battery charge), its battery charge decreases, and its chances of transitioning into a $broken$ state increase (controlled by the outcome B of a Bernoulli trial). Probabilistic rewrite theories unify many different probabilistic models and can express systems involving both probabilistic and nondeterministic features. A more detailed account of probabilistic rewrite theories can be found in [8].

Probabilistic rewrite theories, specified as *system modules* in Maude [5], can be simulated by sampling from probability distributions. This is achieved using Maude's pseudo-random number generator function `random(s)`, with `s` a seed, and a constant `counter` that rewrites to the next natural number using an internal strategy. Using PVeStA [6], randomized simulations generated in this fashion can be used to statistically model check quantitative properties of the system. These properties are specified in a rich, quantitative temporal logic, called Quantitative Temporal Expressions (QuaTEx) [4]. In QuaTEx, real-valued state and path functions are used instead of boolean state and path predicates to quantitatively specify properties about probabilistic models. QuaTEx supports parameterized recursive function declarations, a standard conditional construct, and a *next* modal operator $\bigcirc$, allowing for an expressive language for real-valued temporal properties (Example QuaTEx expressions appear in Section IV). Given a QuaTEx path expression and a Maude module specifying a probabilistic rewrite theory, statistical quantitative analysis is performed by estimating the *expected value* of the path expression against computation paths obtained by Monte Carlo simulations. More details can be found in [4].

B. A Generic Model of DB Protocols

We introduce a model of DB protocols as a probabilistic rewrite theory $\mathcal{R}_{db} = (\Sigma_{db}, E_{db} \cup A_{db}, R_{db})$. Since our aim is to be able to formally model and analyze timing and guessing attacks, for which the rapid bit exchange phase is the most relevant phase of a DB protocol, the other two phases (the setup and verification phases) are only abstractly specified. This keeps the model generic, enabling reasoning

about guessing and timing attacks in different DB protocols. Furthermore, the model is heavily parametrized to capture different attack behaviors and countermeasures, further adding to its generic design. Moreover, by utilizing different facilities provided by its underlying formalism, the model is both probabilistic, specifying randomized behaviors and environment uncertainties, and real-time, capturing time clocks and message transmission delays.

Due to space limitation, we omit some of the details here. The complete specification is available online at https://bitbucket.org/malturki/dbp/.

C. Protocol State

The structure of the model, specified by the MEL subtheory $(\Sigma_{db}, E_{db} \cup A_{db})$ of $\mathcal{R}_{db}$, is based on a representation of actors in rewriting logic, which builds on its object-based modeling facilities. In this model, the state of the protocol is a *configuration* consisting of a multiset of actor objects and messages. Objects communicate asynchronously by message passing. An object is a term of the form `<name: O | AS >`, with `O` the actor object's unique name and `AS` its set of attributes, constructed by an associative and commutative comma (`_,_`) operator. Each attribute is a name-value pair of the form `attr : value`. A message destined for object `O` with payload `C` is represented by a term of the form `O <- C`, where the payload `C` is a term of the sort `Content`.

1) Objects: Two fundamental objects in our model of a DB protocol are the verifier object and the prover object. The verifier object, with actor name `v`, maintains in its state information about the following aspects of the protocol:

Current status. The verifier object maintains a `status` attribute (of sort `Status`) indicating its current execution step in the protocol. The attribute may take on one of eight possible status values: `pending` and `initialized` (for the setup phase), `ready`, `sending`, `sent`, `receiving` and `received` (for the bit-exchange phase), and `completed` (for the verification phase). Furthermore, the verifier object records the number of rounds remaining in the current session of the protocol in the attribute `round`.

Challenge-response. The verifier maintains the two bit strings that are used during the rapid bit exchange phase of the DB protocol using two attributes `list0` and `list1`. Bit strings of sort `BitList` are constructed using an associate empty juxtaposition operator `__`, with identity element `nilCB`. The sort `Bit`, which is declared a subsort of `BitList`, defines a bit as a wrapped 0 or 1 of the form `b(0)` and `b(1)` respectively. The verifier also maintains the values of both the selected challenge bit and the received response bit for the current round in the attributes `challenge` and `response`, respectively.

Round-trip time. To measure the time needed for a challenge-response round to complete, the verifier object maintains two attributes: `mtime-sent`, the measured time value for when the challenge was sent, and `mtime-recv`, the measured time value for when the corresponding response was received. The difference between these two values gives the *measured* round-trip time, and hence the measured distance. Furthermore, to enable modeling absence of in-between-ticks attacks, we also include two attributes for recording the *actual* time values for sending a challenge and receiving the corresponding response: `atime-sent` and `atime-recv`, respectively. The difference gives the *actual* round-trip time. While the measured time is what a realistic verifier having limited processing capabilities would compute, the actual time is computable by an abstract verifier with an extremely powerful and precise processor that is not vulnerable to the in-between-ticks attack (see [13]). All time values are of the sort `Float` to model a dense time domain.

Acceptance and rejection. The decision of acceptance or rejection of a prover in a run of the protocol depends on the acceptance criteria used, which may in turn depend on the measured round-trip time, the correctness of the responses or a combination of those. As we intend for the model to be generic enough to accommodate different acceptance criteria, the decision of acceptance or rejection is not recorded by the verifier object. Instead, the verifier maintains two counters needed for deciding acceptance, which are: (1) the number of rounds the response's measured round-trip time was within the bound (`mtbound-cnt`), and (2) the number of rounds the response was correct (`mbit-cnt`). The desired acceptance criteria can later be defined as part of the quantitative property to be evaluated based on the values of these counters (see Section IV).

The prover object (with actor name `p`) is a much simpler object maintaining a `status` attribute (which can be `pending` or `ready`) and the attributes `list0` and `list1` having the two bit strings used by the protocol.

2) The Scheduler: In addition to objects and messages, the protocol state configuration includes a *scheduler*, which is responsible for managing time and the scheduling of message delivery. The scheduler is a term of the form `{G | L}`, with `G` the current global clock value of the configuration and `L` a time-ordered list (of sort `ScheduleList`) of scheduled messages, where each such message (of sort `ScheduleElem`) is of the form `[T,M]`, representing a message `M` scheduled for processing at time `T`. As time advances, scheduled messages in `L` are delivered (in time-order) to their target objects, and newly produced messages by objects are appropriately scheduled into `L`.

The scheduler is a fundamental component as it serves several purposes. First, it provides a simple mechanism for avoiding unquantified nondeterminism, which is a necessary requirement for the soundness of statistical analysis. The scheduler is used to enforce a strict ordering on message delivery so that at most one message is ready to be consumed at any given moment in time. Additionally, the scheduler enables a simple and elegant solution for managing the global time and the effects of time lapse on the configuration. Finally, the scheduler enables more efficient simulation and analysis by allowing us to specify the granularity of a Monte Carlo simulation of the model (see [11] for further details).

It is important to note that the model uses dense time

(represented by real numbers) to model physical timing of events. Moreover, the (implicit) discrete clock of the verifier object is assumed to always be in sync with the dense physical time of the configuration. In other words, assuming that both physical time and the verifier's clock begin at 0, discrete clock ticks will occur exactly at the positive-integer-valued instants of time. This is to have a model that is faithful to the protocol's description.

D. Model Parameters

To support the analysis of different attack scenarios and countermeasures, the model is designed to be parametric in a number of variables that can be adjusted as needed for the analysis task at hand. The parameters are as follows:

Protocol parameters. Security parameters of the protocol include the number of rounds of rapid bit exchange, maintained by ROUNDS, and the protocol's distance upper bound (measured in time delay), given by MAXRTT. Furthermore, three more parameters capture time delays pertaining to actions made by the verifier: X, the time delay to send out a challenge, and Y and Z, the time delays to record timestamps of a challenge sent and a response received, respectively. Time delays can be non-negative real values or random variables with certain probability distributions. Finally, noise levels in the communication medium are captured by the noise bias parameter, NOISE $\in [0, 1]$, which is the probability of a bit being destroyed while in transit due to noise. Note that noise is (random) disturbance of signals that turns useful information (bits) into distorted signals that carry no useful information. Therefore, unlike Hancke and Kuhn [2], where noise was assumed to simply *flip* bits, which is rather unrealistic, we model a noise-scrambled signal as a constant operator `bN` (which is *not* of the sort `Bit`). Once a bit turns into `bN` due to noise, it cannot be recovered.

Threat model. The in-between-ticks vulnerability can be enabled by setting the Boolean flag IBT?, in which case the verifier does not have access to the actual physical time. In contrast, when IBT? is false, the verifier can use physical time to record the actual timestamps of a challenge sent or a response received, and hence the absence of the vulnerability. Furthermore, the behavior of the prover is defined by the parameter PTYPE, which can take one of three values: 0 for a legitimate prover (no guessing), 1 for a dishonest prover that has access to the protocol session's bit strings (educated guessing), and 2 for an attacker that may not have access to the protocol session's bit strings (random guessing). In the cases of a dishonest prover (when PTYPE > 0), whether the prover is able to mount a guess-ahead attack is determined by the Boolean flag GAHEAD?, and in this case, the parameter GATD specifies how much sooner the attacker/dishonest prover is sending out his premature responses to the verifier. Finally, an upper bound on the (actual) distance between the verifier and the prover is determined by the sum MAXRTT+H, where H is the distance differential, which can in principle have a value in the range $(-\text{MAXRTT}, \infty)$. While an attacker will have a positive value for H (representing being outside the verifier's bound), a legitimate prover is modeled by having H ≤ 0.

Acceptance threshold. Two parameters define acceptance threshold levels for a run of the protocol, one for each acceptance criterion. The parameters are: (1) MIN-MTR, which is the minimum number of successful rounds based on the measured round-trip time, and (2) MIN-MBR, based on the correctness of the response bits. Values may range from 0 up to ROUNDS (which represents the most restrictive acceptance condition requiring all rounds to be successful). Of course, simple majority and large majority can be defined as functions on ROUNDS, and final acceptance can be based on a combination of these parameters (e.g., a large majority for MIN-MTR and a simple majority for MIN-MBR).

We note that using these parameters, different models can be obtained as special instances. For example, the simple model of the Hancke-Kuhn DB Protocol is obtained simply by assuming absence of the in-between-ticks vulnerability (IBT? $=$ `false`), an attacker guessing based on the shared bit sequences (PTYPE $= 1$), no guess-ahead attacks (GAHEAD? $=$ `false`), a positive number of rounds (ROUNDS > 1), acceptance based on both response correctness and measured distance (with MIN-MBR $=$ MIN-MTR $=$ ROUNDS), and, finally, no noise (NOISE $= 0$).

E. Protocol Behavior

The behaviors of the verifier and the prover in the protocol are specified by, possibly conditional and probabilistic, rewrite rules R_{db} in $\mathcal{R}_{db}$. There are ten, semantically rich rewrite rules in R_{db} in total (of which six rules are probabilistic). Due to space limitations, we only highlight below the main transitions showing simplified versions of some representative rules (the full specification is available at https://bitbucket.org/malturki/dbp/). In what follows, we assume the following meta-variable declarations: `iR`, `xR` and `N` of sort `Nat`; `B0`, `B1`, `cB` and `cP` of sort `Bit`; `L0` and `L1` of sort `BList`; `tG`, `tS`, `tR`, `tS` of sort `Float`. We use `AS` for `AttributeSet` and `SL` for `ScheduleList`.

Initiating a round. The beginning of a round is triggered by the verifier object receiving a `beginRound` message, while in the `ready` state.

```
rl [BeginRound] :
<name: v | status: ready, round: iR,
 atime-sent: tS, mtime-sent: tS',
 atime-recv: tR, mtime-recv: tR',
 challenge: cB, AS >
(v <- beginRound) { tG | SL }
=> <name: v | status: sending, round: p(iR),
    atime-sent: (floor(tG) + 1.0 + X),
    mtime-sent: 0.0, atime-recv: 0.0,
    mtime-recv: 0.0,
    challenge: b(if sampleBerWithP(0.5)
                 then 0 else 1 fi), AS >
   mytick(insert({ tG | SL },
   [floor(tG) + 1.0, (v <- sendChallenge)])) .
```

In addition to consuming the message, the rule decrements the number of remaining rounds and resets to zero any stored

time value from the previous round, except the `atime-sent`, which is set to the time value when the challenge will actually be sent. This value is given by the expression `floor(tG) + 1.0 + X`, in which `floor(tG) + 1.0` is the time value when the next discrete clock tick happens and `X` is the random variable for the delay associated with sending out the challenge. Furthermore, the verifier prepares the challenge bit to be used for this round by randomly selecting either the bit 0 or 1 (using an unbiased Bernoulli trial). Finally, the verifier schedules a self-addressed message `v <- sendChallenge` to send out the challenge in the next clock cycle (self-addressed messages are commonly used in actor-based systems to schedule internal events [12]). The operator `mytick` implements an internal mechanism for preparing the next message in the scheduler's queue.

Sending a challenge. Upon receiving this message, the verifier changes its status to `sending` and circularly shifts the bit strings so that the bit-pair for the current round is now at the back of the lists, preparing the bit strings for the next round[1]:

```
rl [Challenge] :
<name: v | status: sending, list0: (B0 L0),
 list1: (B1 L1), atime-sent: tS,
 challenge: cB, AS >
(v <- sendChallenge) { tG | SL }
=> <name: v | status: sent, list0: (L0 B0),
    list1: (L1 B1), atime-sent: tS,
    challenge: cB, AS >
   mytick(insert(insert( { tG | SL } ,
    [ floor(tG) + 1.0, (v <- recordTime) ]),
    [ tS + XDELAY, (p <- challenge(cB)) ])) .
```

Additionally, the verifier schedules two messages: (1) a self-addressed message `v <- recordTime` scheduled to be delivered at the next discrete clock tick for the verifier to timestamp the sending of the challenge, and (2) a challenge message addressed to the prover and scheduled for delivery with a delay `XDELAY` (the transmission delay computed as half the sum MAXRTT + H) beyond the time `tS`, which is the (recorded) actual time for sending the challenge. The challenge message payload carries the challenge bit `cB` previously selected by the verifier.

Recording the timestamp. When the message `v <- recordTime` arrives, the verifier records the timestamp `tG + Y`, where `tG` is the current time value (as given by the scheduler), and `Y` is the random variable associated with recording a timestamp for sending the challenge. There are no new messages scheduled.

Responding to the challenge. When the prover receives the challenge from the verifier, the prover first shifts circularly its stored lists, so that the lists `(B0 L0)` and `(B1 L1)` are rewritten into `(L0 B0)` and `(L1 B1)`. This is to synchronize with the verifier's state and prepare for the next round. The prover then schedules up to two responses, depending on which round is currently being executed and the prover's type. There are generally four distinguishable cases:

1) The protocol run consists of only one round of bit exchange (ROUNDS is 1) or the prover is not attempting to guess-ahead responses (GAHEAD? is `false`). In these cases, the prover simply schedules a 'standard' response to the challenge received to be delivered after a delay given by the transmission delay `XDELAY`.
2) The bit-exchange is multi-round with the current round being the first (so the number of remaining rounds is ROUNDS − 1), and the prover is attempting to guess-ahead responses. In this case, the prover schedules two responses: (1) a standard response to the challenge received delayed by `XDELAY`, and (2) a guess-ahead response in anticipation of the next challenge (the second challenge) to be received later. The latter response is delayed by `(3.0 * XDELAY) - GATD`, which is the amount of time the prover anticipates the verifier to expect the next response, minus the guess-ahead differential given by `GATD`.
3) The bit-exchange is multi-round with the current round *not* being the first or the last round, and the prover is attempting to guess-ahead responses. In this case, the prover does not respond to the received challenge (as it had already responded with a guess in the previous round). Instead, the prover schedules only a guess-ahead response for the next challenge.
4) The bit-exchange is multi-round with the current round being the last (and the prover is guessing-ahead responses). In this case, the prover does not schedule any response, since all the responses needed have already been scheduled before.

The payload of the response message is computed taking into account the value of the challenge bit received, the type of the prover, the current and next bit pairs in the two bit strings, and the noise level. To compute the appropriate payload, two operators are (algebraically) defined: `detBit` and `gssBit`. The former operator determines the response bit based on the challenge received `cB` and the current bit pair `B0` and `B1` if the prover is not guessing responses. Otherwise, if the prover is attempting to guess responses, the operator uses `gssBit` to compute the response.

```
op detBit : Bit Nat Bit Bit -> Bit .
eq detBit(cB, iG, B0, B1)
  = if iG == 0 then    --- No guessing
      if cB == b(0) then B0 else B1 fi
    else               --- Guessing
      gssBit(iG, B0, B1)
    fi .
```

The operator `gssBit` randomly selects between `B0` and `B1` (for educated guessing) or between 0 and 1 (for random guessing) as the guessed response.

```
op gssBit : Nat Bit Bit -> Bit .
eq gssBit(iG, B0, B1)
  = if iG == 1 then    --- Educated guessing
      if sampleBerWithP(0.5) then B0 else B1 fi
```

[1]We note that in the Hancke-Kuhn protocol, unused bits are discarded, but since we are not modeling other types of attacks, e.g. attacks with accelerating clock rates of the prover, this representation works well for our purposes.

```
else                  --- Random guessing
  b(if sampleBerWithP(0.5) then 0 else 1 fi)
fi .
```

We note that in the cases of computing a guessed-ahead response, the next-in-line bit pairs given by `head(L0)` and `head(L1)` are the ones used instead of the current bit pair. Furthermore, in any case, if noise succeeds (based on the outcome of a coin toss with bias given by `NOISE`), the response is simply turned into the illegible signal `bN`.

Receiving the response. When the prover's response message arrives, and the verifier is in the `receiving` state (implying that the verifier is expecting a response), the verifier changes its status to `received`, records the actual time the response was received (which is the current time value given by the scheduler), records the response bit, and schedules a self-addressed message to be delivered by the next discrete clock tick with a time delay given by Z (the random delay associated with recording the timestamp of the response). If the prover's response message arrives while the verifier is not expecting a response (indicated by a status field value different from `receiving`), which can only happen if the prover is mounting a guess-ahead attack, the verifier detects the attack and immediately aborts the protocol session (the status is set to `aborted` and the remaining rounds counter is set to 0).

Recording the timestamp. In this last step of a bit-exchange round, the verifier records its measured timestamp of receiving the response and updates its counters `mbit-cnt` and `mtbound-cnt` appropriately.

IV. Model Checking Timing Attacks

Two fundamental security properties of a DB protocol are *false acceptance* and *false rejection* probabilities. False acceptance occurs when an honest or a dishonest prover outside the allowed perimeter or an attacker is granted access to the protected resource by the (honest) verifier, signaling a successful attack. False rejection, on the other hand, happens when an honest prover is unrightfully denied access to the resource by the (honest) verifier, despite being close enough to the verifier.

To formally analyze false acceptance and false rejection using the rewriting model of $\mathcal{R}_{db}$, we first express acceptance and rejection in a DB protocol as temporal formulas in QuaTEx. The acceptance formula has the following form:

$$\begin{aligned} accepted(t) = \; & \underline{\textbf{if}}\ time() > t\ \underline{\textbf{then}}\ hasSatisfiedThreshold() \\ & \underline{\textbf{else}}\ \bigcirc\, accepted(t)\ \underline{\textbf{fi}}\ ; \\ & \underline{\textbf{eval}}\ \mathbf{E}[accepted(t_0)] \end{aligned}$$

The parameter t is the time limit beyond which protocol execution is halted. In $\mathcal{R}_{db}$, the limit (given by the actual parameter t_0) is set so that the DB protocol is guaranteed to execute all the way to completion (or until the verifier aborts execution). $accepted(t)$ is a recursively defined path expression that uses two state functions: (1) $time()$, which evaluates to the time value of the current state of the protocol (given by the scheduler object), and (2) $hasSatisfiedThreshold()$, which evaluates to `1.0` if the current state of the verifier object indicates that the minimum acceptance threshold has been met, and `0.0` otherwise. Therefore, given an execution path, the path expression $accepted(t)$ evaluates to $hasSatisfiedThreshold()$ in the current state if the protocol run is complete; otherwise, it returns the result of evaluating itself in the next state, denoted by the next-state temporal operator $\bigcirc$. The probability of acceptance can be approximated by estimating the expected value of the formula over random runs of the protocol, denoted by the query $\underline{\textbf{eval}}\ \mathbf{E}[accepted(t_0)]$. Different acceptance criteria, namely measured-time acceptance versus bit-correctness acceptance, give rise to different versions of the formula $accepted(t)$.

The rejection formula $rejected(t)$ is the dual of the acceptance formula and its definition is similar. A statistical approximation of the rejection probability is similarly computed using the query $\underline{\textbf{eval}}\ \mathbf{E}[rejected(t_0)]$.

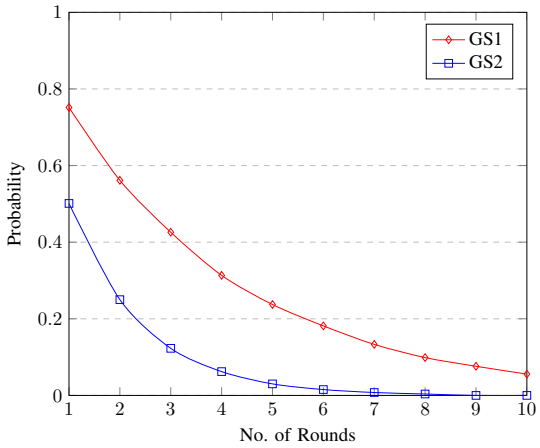
We have used our model given by $\mathcal{R}_{db}$ to statistically model check false acceptance and false rejection formulas under various settings and assumptions using the statistical model checking tool PVeStA. In the analysis presented below, we assume a 99% confidence interval with size at most 0.02. We also fix MAXRTT in $\mathcal{R}_{db}$ to 4.0 time units (similar to [3], [13]) and assume a uniform distribution for the verifier's delays X, Y and Z.

A. Simple Guessing Attacks

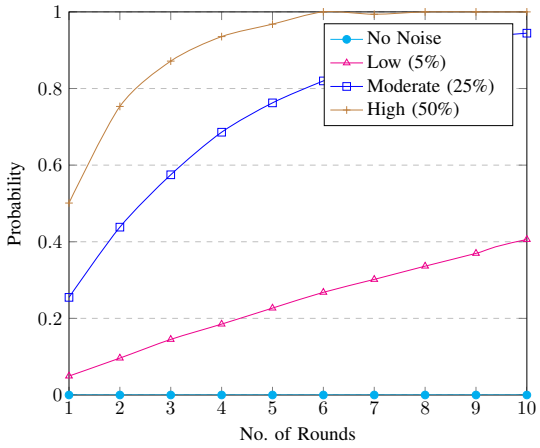
Simple guessing targets acceptance based on correctness of responses. Resilience to simple guessing attacks is a good basic measure of a protocol's resilience to more sophisticated attacks. As mentioned before, there are two possible guessing strategies: educated guessing (denoted by GS1) and random guessing (denoted by GS2). In GS1, the probability of false-acceptance in a single round is 0.75 [2], while for GS2, the probability is only 0.5, which is the optimal false acceptance probability of a DB protocol. To reduce the probability of false-acceptance, the bit-exchange phase is normally run in n rounds, so that for GS1 and GS2, the probabilities are reduced to 0.75^n and 0.5^n, respectively. These results are confirmed by our model for single-round and multi-round runs of the protocol, as the plot in Figure 1(a) shows.

In the presence of noise, to which ultra-wide band communication channels commonly targeted for DB applications are very susceptible, a legitimate prover may be denied access causing false rejections. To investigate the effects of noise, we estimate the probability of false rejection assuming three levels of noise: low (a 5% chance of a bit being destroyed due to noise), moderate (25%) and high (50%). Figure 1(b) plots the estimated probabilities against the number of rounds used. As noise levels increase, the probability of false rejection increases significantly, especially for larger numbers of rounds. Figure 1 demonstrates that reducing false acceptance and false rejections are two conflicting requirements when deciding the number of rounds to be used with noisy channels.

One way to reduce such large false rejection rates is to loosen the acceptance condition so that a prover is accepted

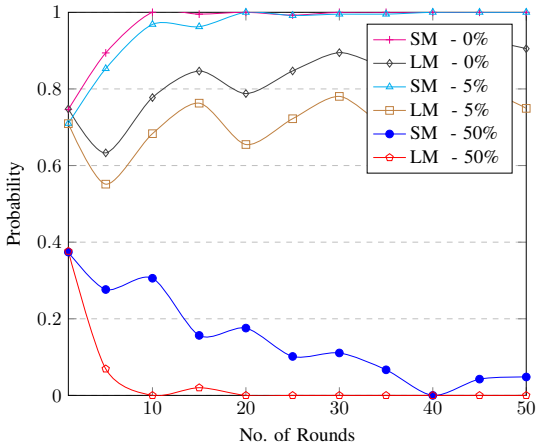


(a) False acceptance with guessing

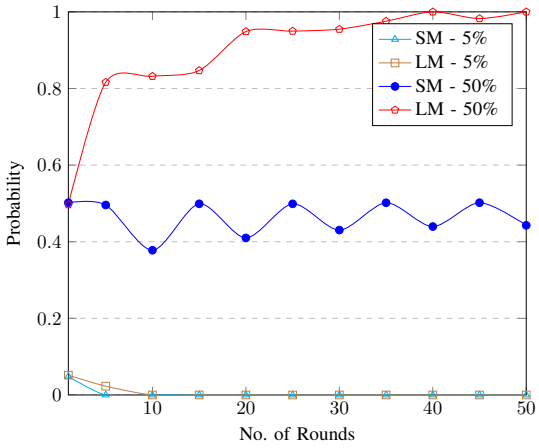


(b) False rejection due to noise

Fig. 1. False acceptance and false rejection based on bit correctness



(a) False acceptance (assuming GS1)



(b) False rejection

Fig. 2. False acceptance and false rejection for simple and large majority

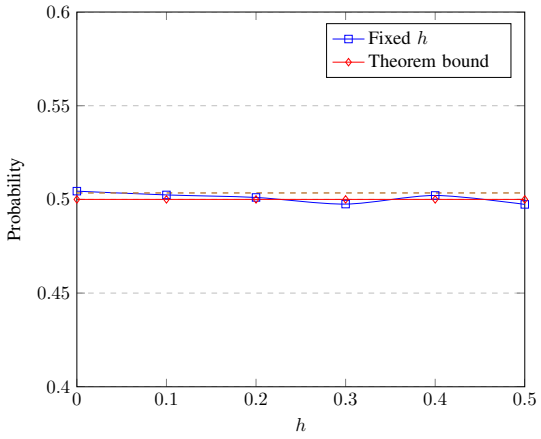
whenever he succeeds in at least $k < n$ rounds (set by the parameter MIN-MBR), where n is the number of rounds. We investigate two acceptance thresholds: the simple majority (SM), where $2k \geq n$, and large majority (LM), where $3k \geq 2n$. Figure 2(a) plots false acceptance probabilities assuming a GS1 attacker, while Figure 2(b) plots false rejection probabilities assuming an honest prover, all at low (5%) and high (50%) noise levels (the zero-noise cases in Figure 2(a) are shown for comparison only).

We first point out the wavy lines, which are characteristic of SM and LM data points and show up in all charts involving SM and LM measurements. This phenomenon is due to the fact that the minimum threshold values for SM and LM acceptance may represent percentages higher than 50% and 67%, respectively, of the total number of rounds, affecting false acceptance and false rejection. For example, in SM, when ROUNDS is 5, the minimum threshold is 3 rounds, representing a 60% acceptance requirement and resulting in a lower false acceptance rate. As the number of rounds increase, however, this phenomenon diminishes as the computed thresholds converge to their target percentages.

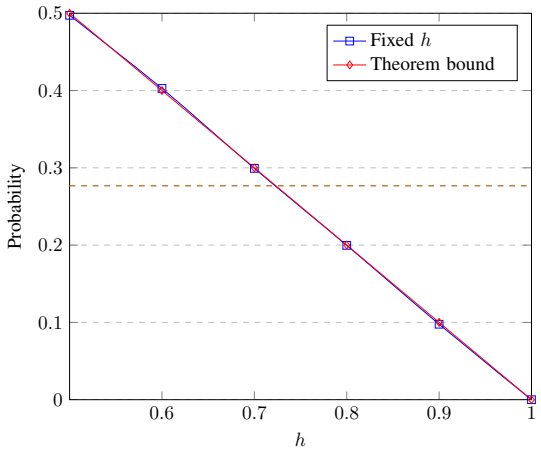
More importantly, we observe from Figure 2(a) that noise does not help simple guessing attacks. In fact, the higher the noise, the less effective the attacks are. Indeed, for an attack round to be successful, the prover must make the correct guess *and* noise will have to fail. The effect of noise is especially prominent when noise is high for large values of ROUNDS, where false acceptance is almost zero for both SM and LM. We also note that for low noise at 5% (and similarly at 0%), false acceptance probabilities are quite high, in the range $0.55 - 1.0$. Although this may seem counter-intuitive at first thought, it should be expected since the acceptance requirements of SM and LM (50% and 67%, respectively) are lower than the success probability of a GS1 prover, which is 75%. Nevertheless, LM performs consistently better than SM in reducing false acceptance across all noise levels. Unfortunately, this comes at the price of increased probability of denying access to legitimate users, as can be seen by comparing the plots in Figure 2a and 2b. SM, however, seems to provide the best compromise for very high levels of noise.

B. *In-between-ticks Attacks*

The in-between-ticks vulnerability requires that the verifier operates on a discrete clock and that the verifier does not



(a) $0 < h \leq 0.5$



(b) $0.5 < h < 1$

Fig. 3. Probability of a breach in the (single-round) DB protocol for different values of h. The dashed lines denote the probability when h is random.

have access to the actual physical timestamps of messages sent or received (IBT? is set to true). Through this vulnerability, an attacker targets acceptance based on the round-trip time (distance) measured by the verifier. To increase his chances of acceptance, the attacker needs to be close enough (but not necessarily too close) to the upper bound stipulated by the protocol (given as MAXRTT in the model). This distance differential is given by h (the parameter H in the model). We note, as mentioned before, that this vulnerability can appear with an honest prover and in the absence of an adversary.

We confirm false acceptance probabilities shown in Theorem 4.3 of [3]. The theorem states that the attacker only needs to be within half-a-clock tick from the bound (so $0 < h \leq 0.5$) to succeed with probability 0.5. Moreover, as the attacker moves away from the bound with $0.5 < h < 1$, the probability of mounting an attack linearly decreases as h increases until it becomes zero for $h \geq 1$. Figure 3 plots false acceptance based on the round-trip time measured by the verifier against the bounds given by Theorem 4.3 of [3], using different values of h ranging from 0 to 1. As the chart shows, the estimated probabilities match very closely those given by the theorem. It is worth noting that the analytical models and machinery used for proving this theorem in [3] are non-trivial and achieving such precise estimations mechanically from our model highlights the effectiveness of the framework.

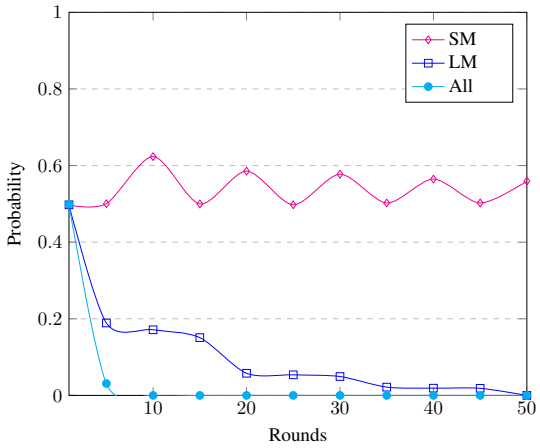
Next, we investigate repeating the bit-exchange round as an attack mitigation measure. The simple majority (SM) and large majority (LM) acceptance thresholds are defined as above. In [13], it was shown that SM has no positive or negative effect on false acceptance when $0 < h \leq 0.5$ but can be effective for $0.5 < h < 1.0$ especially as the number of rounds n increases (Theorem IV.7 in [13]). On the other hand, LM decreases false acceptance significantly as the number of rounds n increases for any positive h (Theorem IV.8 in [13]). These results can be confirmed by our model while gaining a deeper understanding of how these different thresholds compare. Figure 4(a) plots the probability of false acceptance using both acceptance thresholds when the protocol is run for different values of n. For comparison, we also include the most restrictive 'All' acceptance threshold requiring all rounds to pass the bound test successfully. Figure 4(a) shows that, for SM, false acceptance remains close to its single-round value (0.5) as n increases, while LM reduces false acceptance considerably. For $0.5 < h < 1.0$ (not shown in the chart), both acceptance thresholds help mitigate the attack, with LM being more effective at lower values of n (nearly as effective as the 'All' strategy).

Effectiveness against the in-between-ticks attack is one important aspect of a mitigation measure. Another equally important aspect is false rejection. We investigate the performance of these mitigation strategies using the same parameter values used in our analysis of false acceptance above, except now h is negative, representing a legitimate prover within the protocol's bound. Figure 4(b) plots false rejection probabilities for different values of n when $0 > h \geq -0.5$. As n increases, the rejection rate of 'All' increases rapidly all the way to 1.0 while that of SM and LM steadily decreases. Moreover, SM seems to maintain the best false rejection rates across all n with some noticeable margin. For $-0.5 > h > -1.0$ (not shown), false rejection is not an issue as all strategies had zero false rejection probabilities.

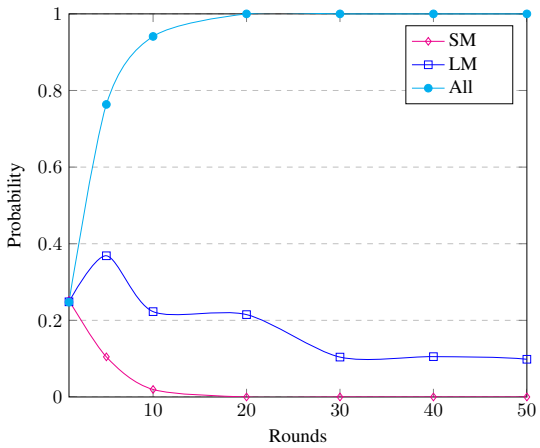
C. Guessing-ahead Attacks

Guessing-ahead attacks target acceptance based on both correctness of the response bits and the calculated time bounds. Consequently, they represent more complicated scenarios than the ones explored earlier in this section. In what follows, we attempt to tame this complexity by assuming that the in-between-ticks vulnerability is not present (IBT? $= \mathit{false}$), and thus, the verifier's time tests are based on the actual physical time (we leave investigating guessing-ahead in the presence of the in-between-ticks vulnerability to future work). It is important to note here that, even with this assumption, the verifier still operates on a discrete clock.

While guessing ahead, the prover needs to maintain synchrony with the verifier's pace of sending out challenges to minimize the chances of being detected. This is important



(a) False acceptance with $0 < h \leq 0.5$



(b) False rejection with $-0.5 < h \leq 0$

Fig. 4. False acceptance and rejection in the multi-round DB protocol

since an unsolicited response is a witness for a guessing-ahead attempt and is used by the verifier to immediately abort the protocol session. To synchronize with the verifier, the prover carefully times a premature response r_{i+1} (for round $i+1$) in relation to the challenge c_i of the preceding round i. Therefore, in an n-round protocol run, the last $n-1$ responses can all be sent out prematurely. The first-round response is not guessed ahead as the prover uses the first challenge to initiate the attack. The diagram in Figure 5 illustrates a possible sequence of challenge-response exchanges with guessing ahead.

In the simplistic, and rather unrealistic, case that a verifier's processing delay between receiving a response and sending out the next-round challenge is *fixed*, say d, an attacker can simply add this delay d to his guess-ahead window to ensure that the attack goes undetected. Even if d is initially unknown, the attacker may use the first two rounds to learn d and then take it into account in subsequent rounds, since it is fixed. However, in reality, d is not fixed as it depends on the verifier's state when the response is received along with the (unpredictable) verifier's hardware delays associated with time-stamping and sending messages. Variability of d is captured to some extent in our rewriting model by having the

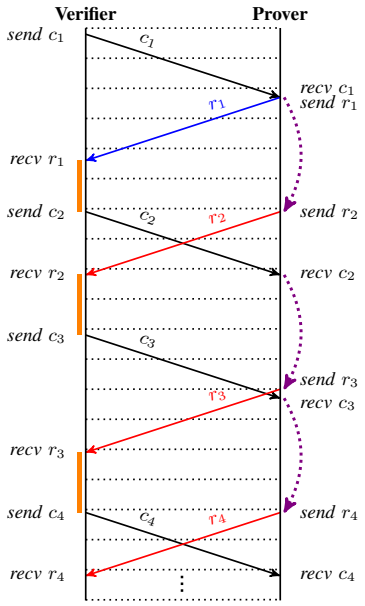


Fig. 5. A challenge-response sequence involving a guessing-ahead prover. The dashed horizontal lines mark the verifier's discrete clock cycle boundaries, and the vertical bars on the left highlight the verifier's (variable) processing delay.

verifier's actions governed by a discrete clock and by modeling the (bounded) random time delay X for sending out challenges. In general, randomizing d across rounds can potentially serve as a countermeasure for guess-ahead attacks [1]. It would therefore be of interest to analyze guessing-ahead attacks in the presence of such randomized verifier behavior.

While the distance bound for a DB protocol is usually public knowledge (given as part of the specification of the protocol), the exact location of the verifier may not necessarily be known. This means that a prover attempting a guess-ahead attack may not know exactly what his distance h from the bound is, although he may still be aware of the verifier's approximate location. Therefore, we define the following guess-ahead strategies and consider them in our analysis:

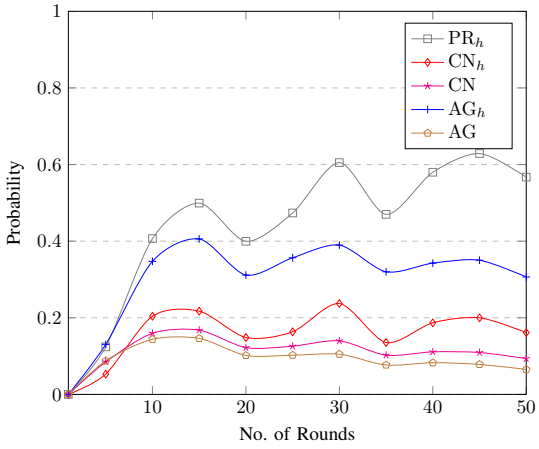
1) The *prescient strategy* (PR_h), which represents a prover who knows the verifier's exact location (and hence h) and, therefore, responds exactly $2h$ time units sooner than the anticipated time of receiving the challenge. It models a strong attacker and serves as a benchmark for the other attack strategies' performance.
2) The *conservative strategy* (CN_h), representing a prover who is uncertain about the exact location of the verifier and who aims to minimize the probability of the attack being detected. In this strategy, the guess-ahead time is chosen uniformly at random from $[h, 2h]$. A variation of this strategy, denoted CN, draws its guess-ahead time from the window $[T/4, T/2]$, where T is the verifier's clock cycle period. CN can be used if the uncertainty about the verifiers' location is very high, and yet the prover needs to be conservative in timing the attack.
3) The *aggressive strategy* (AG_h), uses more aggressive guess-ahead timing (chosen uniformly from

$[3h/2, 5h/2]$), hoping to increase his chances of meeting the time bound requirement while risking being detected. As above, the variant AG, in which the guess-ahead time is drawn from $[T/2, T]$, models an aggressive strategy with unknown h.

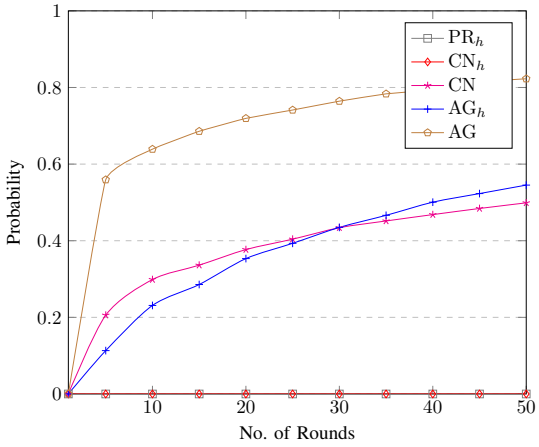
We consider these three strategies (and their variants) in our analysis. We show below the results of the most interesting cases, where we have a capable attacker (GS1) who is fairly close to the distance bound ($0 < h \leq 0.5$) and a somewhat cautious verifier using the large majority for both bit correctness and time tests with a typical level of noise (5%). Other possibilities can also be analyzed but they are not expected to yield new or interesting results.

Figure 6(a) plots the probability of a successful attack using these attack strategies for different rounds. We first observe that knowing something about h can significantly increase an attacker's chances of success. The prescient strategy PR_h can achieve success probabilities ranging from 40% to 60%, even at high round counts (and in the presence of noise!). Even if the exact value of h is unknown, the strategy AG_h can achieve 30% to 40% success, which is still quite high, and it outperforms the conservative strategy CN_h. However, knowing very little about the location of the verifier severely limits the effectiveness of the attack (CN and AG). Furthermore, in this case, following an aggressive strategy (AG) is counterproductive.

There are four ways in which a guessing-ahead round may fail: (1) the response bit is incorrect, (2) the response arrives too late, (3) the response gets destroyed by noise, or (4) the response arrives too early (the attack is detected). We have looked into the first three already. It would therefore be insightful to investigate round failures due to the attack being detected by the verifier. Figure 6(b) plots the attack detection probability for the different attack strategies. While PR_h and CN_h are never detected by the verifier, the attack detection rates for the other strategies increase with the number of rounds, with the blindly aggressive strategy AG being the worst performer. AG_h, however, maintains a detection rate that is comparable to the much more conservative strategy CN, despite being much more effective than CN (as Figure 6(a) shows). By knowing approximately where the verifier is located, an aggressive guessing-ahead strategy can be quite effective.



(a) False acceptance



(b) Attack detection rate

Fig. 6. False acceptance and attack detection with guessing-ahead (GS1) and low noise

V. Related Work

A formal model of physical security protocols that extends the Dolev-Yao model with dense time, network topology, and node location is presented in [14], [15]. The model is formalized in Isabelle/HOL and used to verify distance bounding protocols where the concrete message theory includes exclusive-or. In [16] the model is used for additional protocols including formally analyzing ranging, distance bounding, and secure time synchronization.

In [17], the Dolev-Yao algebraic method for protocol analysis is refined by a probabilistic model of guessing, needed to analyze protocols that mix weak cryptography with physical properties of nonstandard communication channels. The model is used to develop a precise security proof for a proximity authentication protocol, due to Hancke and Kuhn.

Attacks that can be found by models of cyber-physical security protocols using dense time, but not when using discrete time are presented in [3]. This is illustrated with the in-between-ticks, which can be carried out on most DB protocols. A probabilistic analysis of the attack is also presented.

A unified framework is introduced in [18] that aims to improve analysis and design of DB protocols. The framework characterizes different attacks, adversary/prover capabilities and strategies. It introduces notions of black-box and white-box models, and the relation between the different attacks with respect to these models. The framework is demonstrated with a detailed analysis of the Munilla-Peinado DB protocol. The framework is not formalized and analysis is carried out by hand.

An exhaustive classification for attacks on DB protocols is defined in [19] that includes a new attack called Distance Hijacking Attack. Countermeasures for several attacks are proposed. The formal framework of [14], [15] is extended

to support reasoning about overshadowing attacks and the resulting framework is used to prove the absence of attacks after the proposed countermeasures are applied. However, all considered attacks are caused by failures in the authentication phase of DB protocols, not by failures of the distance measurement phase of DB protocols that we consider in our analysis here.

The approach followed in this work, which is based on rewriting logic and MAUDE, has been used before to formally verify quantitative properties of probabilistic systems, including analysis of resilience against DoS attacks [11], [20], analysis and redesign of wireless sensor networks [21], evaluation of design alternatives of distributed transaction systems [22], among several others.

VI. CONCLUSION

We presented a framework based on rewriting logic through which DB protocols can be faithfully specified and analyzed. The framework is characterized by being both expressive and generic, capturing the probabilistic, real-time interactions of various instances of a DB protocol. Using MAUDE and PVESTA, probabilistic properties of false acceptance and false-rejection under different settings and attacker models were mechanically verified through statistical model checking. Verification confirmed very precisely some known results (for which complex manual proofs had to be developed) and enabled the evaluation of other settings (such as noisy channels) and attacker strategies (including guessing-ahead strategies) that are not easily manually analyzable.

There are several possible extensions for future research. As guessing-ahead and in-between-ticks attacks are both forms of timing attacks, an investigation of their interplay is needed: Can an attacker amplify his chances of distance fraud by mounting a combination of guessing and in-between-ticks attacks? If so, what strategies would be most effective and under what assumptions? Moreover, what countermeasures are most effective in preventing combinations of both attacks and how do they affect the protocol's false-rejection rate? These and other properties, such as considering probability distributions for the verifier's delays X, Y and Z other than the uniform distribution, can quickly become too complex for manual analysis, but they can still be formally and automatically verified using statistical model checking. Another direction for future research is to extend our model and framework to allow investigating other kinds of attacks on DB protocols that may involve parties other than the verifier and the prover, such as distance hijacking attacks. A third direction is to develop suitable probabilistic extensions to timed multiset rewriting in which DB protocols (and other probabilistic timed systems) can be specified and probabilistically analyzed.

ACKNOWLEDGMENT

Alturki is partially supported by KFUPM through his sabbatical project SL161003. Ban Kirigin is supported in part by the Croatian Science Foundation under the project UIP-05-2017-9219. Scedrov is partially supported by ONR. The participation of Kanovich and Scedrov in the preparation of this article was partially within the framework of the Basic Research Program at the National Research University Higher School of Economics (HSE) and supported within the framework of a subsidy by the Russian Academic Excellence Project '5-100'. Talcott is partly supported by ONR grant N00014-15-1-2202 and NRL grant N0017317-1-G002.

REFERENCES

[1] S. Brands and D. Chaum, "Distance-bounding protocols," in *Advances in Cryptology — EUROCRYPT '93: Workshop on the Theory and Application of Cryptographic Techniques Lofthus, Norway, May 23–27, 1993 Proceedings*, T. Helleseth, Ed. Berlin, Heidelberg: Springer, 1994, pp. 344–359.

[2] G. P. Hancke and M. G. Kuhn, "An RFID distance bounding protocol," in *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05)*, Sept 2005, pp. 67–73.

[3] M. Kanovich, T. Ban Kirigin, V. Nigam, A. Scedrov, and C. Talcott, "Time, computational complexity, and probability in the analysis of distance-bounding protocols," *Journal of Computer Security*, vol. 25, no. 6, pp. 585–630, 2017. [Online]. Available: https://doi.org/10.3233/JCS-0560

[4] G. Agha, J. Meseguer, and K. Sen, "PMaude: Rewrite-based specification language for probabilistic object systems," *Electronic Notes in Theoretical Computer Science*, vol. 153, no. 2, pp. 213–239, 2006.

[5] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott, *All About Maude - A High-Performance Logical Framework*, ser. Lecture Notes in Computer Science. Secaucus, NJ, USA: Springer-Verlag, 2007, vol. 4350.

[6] M. A. Alturki and J. Meseguer, "PVeStA: A parallel statistical model checking and quantitative analysis tool," in *Algebra and Coalgebra in Computer Science*, ser. Lecture Notes in Computer Science, A. Corradini, B. Klin, and C. Cîrstea, Eds. Springer Berlin / Heidelberg, 2011, vol. 6859, pp. 386–392.

[7] J. Meseguer, "Conditional rewriting logic as a unified model of concurrency," *Theor. Comput. Sci.*, vol. 96, no. 1, pp. 73–155, 1992.

[8] K. Sen, N. Kumar, J. Meseguer, and G. Agha, "Probabilistic rewrite theories: Unifying models, logics and tools," University of Illinois at Urbana Champaign, Tech. Rep. UIUCDCS-R-2003-2347, May 2003.

[9] J. Meseguer, "Membership algebra as a logical framework for equational specification," in *Proc. WADT'97*, ser. Lecture Notes in Computer Science, F. Parisi-Presicce, Ed., vol. 1376. Springer, 1998, pp. 18–61.

[10] R. Bruni and J. Meseguer, "Semantic foundations for generalized rewrite theories," *Theor. Comput. Sci.*, vol. 360, no. 1-3, pp. 386–414, 2006.

[11] G. Agha, C. A. Gunter, M. Greenwald, S. Khanna, J. Meseguer, K. Sen, and P. Thati, "Formal modeling and analysis of DoS using probabilistic rewrite theories," in *International Workshop on Foundations of Computer Security (FCS'05)*. Chicago, IL: IEEE, June 2005.

[12] G. Agha, *Actors: a model of concurrent computation in distributed systems*. Cambridge, MA, USA: MIT Press, 1986.

[13] M. Kanovich, T. Ban Kirigin, V. Nigam, A. Scedrov, and C. Talcott, "Can we mitigate the attacks on distance-bounding protocols by using challenge-response rounds repeatedly?" in *Workshop on Foundations of Computer Security*, June 2016.

[14] D. Basin, S. Capkun, P. Schaller, and B. Schmidt, "Let's get physical: Models and methods for real-world security protocols," in *Theorem Proving in Higher Order Logics: 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–22.

[15] P. Schaller, B. Schmidt, D. Basin, and S. Capkun, "Modeling and verifying physical properties of security protocols for wireless networks," in *2009 22nd IEEE Computer Security Foundations Symposium*, July 2009, pp. 109–123.

[16] D. Basin, S. Capkun, P. Schaller, and B. Schmidt, "Formal reasoning about physical properties of security protocols," *ACM Transactions on Information and System Security*, vol. 14, no. 2, 2011.

[17] D. Pavlovic and C. Meadows, "Bayesian authentication: Quantifying security of the Hancke-Kuhn protocol," *Electronic Notes in Theoretical Computer Science*, vol. 265, no. Supplement C, pp. 97 – 122, 2010, proceedings of the 26th Conference on the Mathematical Foundations of Programming Semantics (MFPS 2010). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1571066110000861

[18] G. Avoine, M. A. Bingöl, S. Kardaş, C. Lauradoux, and B. Martin, "A framework for analyzing RFID distance bounding protocols," *J. Comput. Secur.*, vol. 19, no. 2, pp. 289–317, Apr. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1971859.1971864

[19] C. Cremers, K. B. Rasmussen, B. Schmidt, and S. Capkun, "Distance hijacking attacks on distance bounding protocols," in *2012 IEEE Symposium on Security and Privacy*, May 2012, pp. 113–127.

[20] M. Alturki, J. Meseguer, and C. A. Gunter, "Probabilistic modeling and analysis of DoS protection for the ASV protocol," *Electron. Notes Theor. Comput. Sci.*, vol. 234, pp. 3–18, 2009.

[21] M. Katelman, J. Meseguer, and J. Hou, "Redesign of the LMST wireless sensor protocol through formal modeling and statistical model checking," in *Proc. of FMOODS '08*, ser. Lecture Notes in Computer Science, vol. 5051. Berlin, Heidelberg: Springer, 2008, pp. 150–169.

[22] S. Liu, P. C. Ölveczky, J. Ganhotra, I. Gupta, and J. Meseguer, "Exploring design alternatives for RAMP transactions through statistical model checking," in *Formal Methods and Software Engineering: 19th International Conference on Formal Engineering Methods, ICFEM 2017, Xi'an, China, November 13-17, 2017, Proceedings*, Z. Duan and L. Ong, Eds. Cham: Springer International Publishing, 2017, pp. 298–314.