Automatic analysis of distance bounding protocols

Sreekanth Malladi[1], Bezawada Bruhadeshwar[2], and Kishore Kothapalli[2]

[1] Dakota State University
Madison, SD 57042, USA
Sreekanth.Malladi@dsu.edu
[2] International Institute of Information Technology
Hyderabad, India
{bbruhadeshwar,kkothapalli}@mail.iiit.ac.in

Abstract. Distance bounding protocols are used by nodes in wireless networks for the crucial purpose of estimating their distances to other nodes. Past efforts to analyze these protocols have only been manual. In this paper, we use the constraint solver tool to automatically analyze distance bounding protocols: We first formulate a new trace property called *Secure Distance Bounding* (SDB) that protocol executions must satisfy. We then classify the scenarios in which these protocols can operate considering the (dis)honesty of nodes and location of the attacker in the network. Finally, we extend the constraint solver tool so that it can be used to test protocols for violations of SDB in those scenarios and illustrate our technique on several examples.

Keywords. Wireless networks, Sensor networks, Localization, Distance bounding, Formal methods, Constraint solving, Cryptographic protocols, Timed analysis.

1 Introduction

A *distance bounding (DB) protocol* is used by a "verifier" node in wireless networks to calculate an upper bound on the distance to a "prover" node in the network. Distance bounding helps in crucial applications such as localization, location discovery and time synchronization. Hence, the security of DB protocols is an important and critical problem.

As an example of a DB protocol, consider a simple extension of the Echo protocol (Fig. 1.a) presented in [12]. In the figure, V is the verifier, P is the prover; N_V is a nonce; $Sig_{pk(P)}([N_V,\ V,\ P])$ is the signature of P to be verified with it's public-key, denoted $pk(P)$. Let t_i be the time on the clock when event i occurs. Then, V can calculate the bound 'd' on the distance to P as: $d = \frac{(t_4 - t_1)\ -\ (t_3 - t_2)}{2} \times s$, where '$s$' is the speed of the signal.

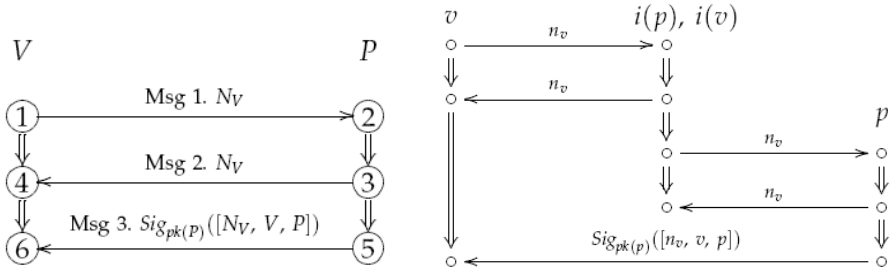


Fig. 1. (a) Extended Echo protocol **P1** (b) Man-in-the-Middle Attack on **P1**

In the presence of attackers, DB protocols can fail to achieve their main goal of establishing a valid distance bound. For instance, the above protocol has a possible attack wherein an attacker i plays Man-In-The-Middle and succeeds in showing p as being closer to v[3] than it really is (Fig. 1.b).

Analysis of DB protocols involves examining whether it is possible to make a party appear closer than it really is, to an honest verifier. The problem is different and difficult compared to standard Dolev-Yao analysis of protocols that only consider whether an attacker can generate messages required to violate some security property. Here, we need to factor in the time required for message generation as well, which can vary based on the input size and cryptographic parameters. Automated analysis is much desired, given the problems and distrust in manual analysis of protocols that have been reported in literature [6]. There have been numerous instances when automated techniques found attacks on protocols that manual, hand-based techniques could not (e.g. [7, 9, 11]).

Past work. The few published efforts to analyze DB protocols have been largely incomplete: The classical work of Brands and Chaum [2] is mostly informal and specific to the protocols introduced in that paper. Sastry et al. [12] show that in their "Echo" protocol, the prover cannot respond before receiving the verifier's nonce but the protocol is used only for "in-range" verification and also too simple without any authentication. Meadows et al. [10] give a method to analyze both distance bounding and authentication aspects, but the method like the previous two, is manual, not automated.

Our contribution. To address these concerns, we will show a method to automatically analyze DB protocols using the constraint solving technique of Millen-Shmatikov. Our method is based on formal modeling of timed protocols and distance bounding properties. Further, it is fully automated with minor changes to the existing constraint solver[4]. Some highlights of our contribution are:

[3] We use lower case for v and p now since we are referring to the protocol execution.

[4] on-line demo at `http://homepages.dsu.edu/malladis/research/ConSolv/Webpage/`

1. Like many past strand space extensions, our formal modeling and framework give a simple, clean and useful geometric flavor to the study of DB protocols that could be used or extended to many other studies such as localization algorithms;
2. Some properties we prove about DB protocols allow the use of conventional Dolev-Yao style analysis, completely eliminating the need to consider the more complicated timing aspects. This is useful when it is difficult to extend existing methods for conventional key establishment protocols to analyze or verify DB protocols (e.g. ProVerif [1]).

Organization. We will first develop a timed protocol model extending strand spaces in Section 2. We will then explain how constraint solving can be used to generate timed protocol executions in Section 3. We will formalize secure distance bounding and explain our technique to detect violations for it in Section 4. We will identify the scenarios under which DB protocols need to be analyzed in Section 5. We will illustrate our analysis approach on some examples in Section 6, and conclude with a discussion of future and related works.

2 Protocol model - Timed strand spaces

Our protocol model is based on the strand space model of [15] extended with the introduction of a new field, "time" for labels on nodes. This field is used to represent the current time on the clock at the node for an agent.

Definition 1. [**Node**] *A* node *is a 3-tuple with fields* time, sign, *and* term. *Time is the current time on the clock, sign can be* $+$ *or* $-$ *denoting "send" and "receive" respectively and term to be defined next.*

We will describe how to populate the times on nodes partly in this section and partly in the next section. We consider protocols in which messages are constructed using a free term algebra:

Definition 2. [**Term**] *A* term *is one of the following:* Variable *(can be of types Agent, Nonce etc.);* Constant *(numbers 1, 2, ...; name of the attacker* ϵ *etc.);* Atom*;* Pair *denoted* $[t_1, t_2]$ *if* t_1 *and* t_2 *are terms;* Public-Key *denoted* $pk(A)$ *with* A *of type Agent;* Shared-Key *denoted* $sh(A, B)$ *with* A *and* B *of type Agent;* Asymmetric encryption *denoted* $[t]^{\rightarrow}_{k}$ *where* t *and* k *are terms;* Symmetric encryption *denoted* $[t]^{\leftrightarrow}_{k}$ *where* t *and* k *are terms;* Hash *denoted* $h(t)$ *where* t *is a term;* Signature *of a term* t *denoted* $Sig_{pk(A)}(t)$ *to be validated using* $pk(A)$.

A "ground" term is any term with no variables in it. We will drop the superscript $\rightarrow$ or $\leftrightarrow$ if the mode of encryption is contextually either obvious or irrelevant.

Definition 3. **[Subterm]** *Term* t *is a* subterm *of* t' *(i.e.* $t \sqsubset t'$*) if* $t = t'$*, or if* $t' = [t_1, t_2]$ *with* $t \sqsubset t_1 \vee t \sqsubset t_2$*, or if* $t' = [t^{''}]_{k'}$ *with* $t \sqsubset t^{''}$*, or if* $t' = h(t^{''})$ *with* $t \sqsubset t^{''}$*, or if* $t' = Sig_{pk(A)}(t^{''})$ *with* $t \sqsubset t^{''}$*. Term* t *is a* proper subterm *of* t' *if* $(t \sqsubset t') \wedge (t \neq t')$*.*

Strands capture roles of a protocol.

Definition 4. **[Strand]** *A* strand *is a sequence of nodes. For instance* $s = \langle n_1, \ldots, n_m \rangle$ *is a strand with* m *nodes. Nodes in a strand are related by the edge* $\Rightarrow$ *defined such that if* n_i *and* n_{i+1} *belong to the same strand, then* $n_i \Rightarrow n_{i+1}$*. A* parametric strand *is a strand with no atoms in the terms on its nodes.*

Protocol roles are modeled as partially instantiated parametric strands that we name *semi-strands* where messages contain variables and atoms depending on the knowledge of agents concerning message subparts. For instance, the verifier strand of the protocol presented in the Introduction is represented as

$$\langle \quad +[0,\ n_v], \quad -[T_4,\ n_v], \quad -[T_6,\ Sig_{pk(P)}([n_v,\ v,\ P])] \quad \rangle$$

Notice that the first node starts at time '0' which is not a universal '0' but a local start time for the agent who dons this strand. Also notice that the times on the other two nodes T_4 and T_6 are not fixed. The rationale for this is to be explained shortly.

A set of semi-strands is called a *semi-bundle.* We will say that term t belongs to a semi-bundle S (i.e. $t \in S$) if $(t = \text{term}(n))$ for some $(n \in s)$ and $(s \in S)$.

A *bundle* is a possible protocol execution obtained by consistently instantiating all the variables in the semi-bundle and using $\rightarrow$ edges between nodes on different strands.

Definition 5. **[Bundle]** *A* bundle *is a collection of strands and an acyclic digraph defined on a mapping of nodes to edges* $\rightarrow$ *and* $\Rightarrow$ *such that if node* n_i *sends a message that* n_j *receives, then* n_i*,* n_j *are related by the edge* $\rightarrow$ *(denoted* $n_i \rightarrow n_j$*). Further, if there is a node* n *in the bundle that receives a term* t*, then there is another node* m *in the bundle, that sends* t *such that* $m \rightarrow n$*.*

Note that this bundle is a 3-dimensional graph with strands located vertically anywhere in the cube. Nodes in a bundle are also related by precedence relation denoted $\preceq$ which is a partial order:

Definition 6. **[Precedes]** *The relation* $\preceq$ *is defined such that if nodes* n_i*,* n_j *exist in a bundle* $\mathcal{C}$*, then* $n_i \preceq n_j$ *if they are on the same strand with* $i \leq j$*; further,* $n_i \prec n_j$ *if* $n_i \rightarrow n_j$*.*

We will use $\preceq$ on stand-alone strands in semi-bundles as well: Let s be a strand in a semi-bundle S. Then, $(\forall n_i, n_j \in s)(s \in S)(i \leq j \Rightarrow n_i \preceq n_j)$.

We do not include the notion of penetrator strands as in the classical strand spaces formalism of [15]. Rather, we consider a single penetrator also modeled as a single strand that captures all the "penetrator actions" in the bundle defined as below:

Definition 7. **[Penetrator action]** *A* penetrator action *is a sequence of edges* $t_1 \rightarrow t_2 \Rightarrow t_3 \rightarrow t_4$ *where* $t_2 \Rightarrow t_3$ *is an edge on the penetrator strand.*

The idea is that the single $\Rightarrow$ edge in a penetrator action represents all the penetrator strands in the classical model of [15] to generate the term to be sent. Multiple penetrators could be added in the 3-dimensional cube if desired, although we only consider a single "Machiavellian" attacker with full control of the network in the spirit of [14][5].

Next we define the "elapsed time" between any two nodes n_i, n_j in a bundle C with $n_i \preceq n_j$ using the notion of *weights* and *paths*:

Definition 8. **[Weight or Elapsed time]** *The* weight *of an edge is the (absolute) difference in times between the nodes that are connected by the edge. A* path *is a sequence of nodes such that every node in the sequence is related to the subsequent node by a* $\rightarrow$ *or a* $\Rightarrow$*. The weight of a path is the sum of the weights of all the edges in the path.*

We will denote the path between n_i and n_j as (n_i, n_j) when there is only one route between n_i, and n_j.

The weight of a $\pm t \Rightarrow +t'$ edge should be preset and constant for each semi-strand. In the case of penetrator strand, those weights should be calculated using penetrator actions required to generate the $+$ node. On the other hand, the weight of a $\pm t \Rightarrow -t'$ edge cannot be fixed since an agent can only know the length of time after which it sends a message, but cannot always predict when it might receive a message from another agent, accurately.

Weights of $\rightarrow$ edges indicate the time of traversal for messages which depends on the message length, distance and the velocity of the signal. We assume that there is an appropriate formula for an environment to calculate the weight of these edges, using those parameters.

Definition 9. **[Relay, Simple relay]** *A* relay *is a penetrator action* $+t \rightarrow -t \Rightarrow +t \rightarrow -t$*. A* simple relay *is a relay with the weight of the* $\Rightarrow$ *edge being zero.*

We develop the notion of "ideal" and "real" bundles to distinguish protocol executions where the penetrator plays a passive role of merely observing message exchanges between agents with those where she plays an active role of faking and changing messages.

[5] This might be unrealistic in wireless networks, but the stronger model allows us to find all attacks including those under weaker attackers.

Definition 10. [**Ideal and Real bundles**] *An* ideal bundle *B for a protocol P is a bundle formed from a semi-bundle S with exactly one semi-strand per parametric strand of P where every penetrator action is a simple relay $+\sigma t \rightarrow -\sigma t \Rightarrow +\sigma t \rightarrow -\sigma t$ for some substitution σ such that $(\forall s \in S)((\exists s' \in B)(s' = \sigma s))$. A* real bundle *is any bundle from any other semi-bundle from P.*

3 Extending constraint solving to find elapsed time

We will now extend the constraint solving technique of [11] to give a "recipe" to produce the timed bundles defined in Section 2 including honest strands and the single penetrator strand with all the penetrator actions.

The previous section only noted that weights of $\pm \Rightarrow +$ edges should be preset; this section will complete labeling of nodes since weights on $\pm \Rightarrow -$ edges are calculated dynamically setting the times on '$-$' nodes during protocol executions. The elapsed time between any two nodes in such bundles can then be calculated by summing up the weights on all the edges in the path between the nodes.

Constraint solving is a procedure to determine if a semi-bundle is completable to a bundle using a substitution to variables. A constraint sequence is first drawn from node interleavings of the semi-bundle indicating that '$-$' nodes should be derivable by the attacker with his actions and terms on all prior '$+$' nodes.

Definition 11. [**Constraint sequence**] *A* constraint sequence *$C = \langle\ \mathrm{term}(n_1) : T_1,\ \ldots,\ \mathrm{term}(n_k) : T_k\ \rangle$ is from a semi-bundle S with k '$-$' nodes if $(\forall\ n,\ n')(\ ((\mathrm{term}(n') : T \in C) \wedge (\mathrm{term}(n) \in T)) \Rightarrow (n \preceq n')\ )$. Further, if $i < j$ and n_i, n_j belong to the same strand, then $n_i \preceq n_j$ and $(\forall i)(T_i \subseteq T_{i+1})$.*

We consider a set of attacker operators Φ and an infinite set of terms that can built using Φ on a finite set of terms T, denoted $\mathcal{F}(T)$. Although our techniques in this paper are largely independent of the kind of operators in Φ, we will consider that they represent the standard Dolev-Yao attacker as defined in [11].

The possibility of forming bundles from a given semi-bundle can be determined by testing if constraint sequences from it are satisfiable:

Definition 12. [**Satisfiability, Realizability**] *A constraint is $m : T$ is* satisfiable *under a substitution σ if $\sigma m \in \mathcal{F}(T)$. A constraint sequence is* satisfiable *with σ, denoted $\sigma \vdash C$ if $(\forall m : T \in C)(\sigma m \in \mathcal{F}(\sigma T))$. A '$-$' node is* realizable *if the corresponding constraint is satisfiable. A semi-bundle is* completable *to a bundle if a constraint sequence from it is satisfiable.*

Millen-Shmatikov have shown a constraint satisfaction procedure, denoted **P** that is terminating, sound and complete wrt Φ and $\mathcal{F}$. **P** applies a

set of symbolic reduction rules R[6] to each constraint, in order to reduce it them to "simple constraints" (with only a variable each on the left side).

We will consider that each reduction rule in R corresponds to an attacker action and we will calculate the weights of $\Rightarrow$ edges of a bundle to be the sum of the times taken by each rule.

In an expanded version of this paper ([8]) we also give an algorithm denoted **PB** that produces timed bundles as defined in Section 2, using **P** to calculate the weights of $\Rightarrow$ edges. We illustrate **PB** in that paper on the NSPK/NSL protocols. Further, we prove that **PB** terminates, and is sound and complete.

4 Analyzing DB protocols

We will now formalize secure distance bounding using the concept of ideal and real bundles defined in Section 2.

4.1 Formalizing Secure Distance Bounding

A DB protocol is used by a verifier v to establish an upper bound on the distance to a prover p. Ideally, if the following assumptions hold: **(a)** The positions of v and p are fixed, **(b)** The intervals between creating and sending messages are fixed, **(c)** v, p are honest and **(d)** There is no attacker; then there indeed exists an upper bound on the distance that can be calculated by calculating the elapsed time between two nodes *Request* and *Response* on v with *Request* a send node, *Response* a receive node and $\mathit{Request} \prec \mathit{Response}$ as explained in Section 2.

We will call the nodes between *Request* and *Response* in the verifier strand of a DB protocol as the "DB part" and the other nodes as the "authentication part". Further, we will use the term "Time of Flight" or its abbreviation as ToF to refer to the elapsed time between *Request* and *Response*.

Now the upper bound that is calculated by v can be lowered compared to the one obtained under ideal conditions, if **(a)** the ToF between its *Request* and *Response* is lowered *and* **(b)** if v is sent all the messages in the protocol that it expects to receive from p.

This is the main insight in defining secure distance bounding as a trace property: We first calculate the ToF under ideal conditions and check whether a "real" execution of the protocol in the presence of the penetrator can result in a calculation of ToF that is lower than the ideal value. Note that we assume weights of $\pm m \Rightarrow +m'$ edges are set for strands in semi-bundles, by following the same measures to calculate time taken for message construction outlined in Section 3.

Definition 13. **[Secure Distance Bounding (SDB)]** *Let t and t' be the elapsed times in the verifier strand of an ideal and real bundle (B and B')*

[6] We refer the reader to [11] for details on **P** and R.

respectively from a semi-bundle S, *between the Request and Response nodes. Then* Secure Distance Bounding (SDB) *is satisfied in* B', *whenever* $t < t'$. *Conversely, SDB is violated in* B' *if* $t > t'$.

This definition is dependent on what we consider an ideal bundle to be. In Section 2, we defined it to be one with no penetrator actions, but when the penetrator is further from v than p is, we would need to make the bundle between the penetrator and v as the ideal. More on this is explained in Section 5.2.

5 Protocol execution scenarios

Before explaining our technique to test protocols for violations of SDB, we will consider the scenarios under which a DB protocol can operate.

5.1 Scenarios based on honesty

We first consider scenarios in which the prover is honest or dishonest.

Scenario A (honest prover). With the verifier, honest prover and an attacker, this scenario captures MITM/Mafia attacks [4]. The attack described in the Introduction is one such attack.

Scenario B (dishonest, colluding prover). With the verifier, dishonest prover and attacker, this scenario captures terrorist/collusion attacks [4]. Here, the prover colludes with an attacker who is presumably closer to the verifier, by passing some or all of its information including secret keys and messages (partial or full collusion). The protocol in the Introduction is vulnerable to such an attack (Fig. 2.a).

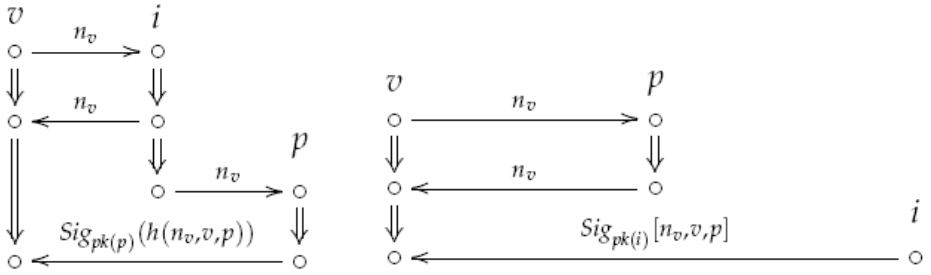


Fig. 2. (a) Scenario B — colluding attacker (b) Scenario 2 — further attacker

5.2 Scenarios based on location of attacker

Independent of the honesty of agents, we should also categorize protocol execution scenarios based on the location of the attacker in the network with respect to the verifier and the prover.

Scenario 1 (closer attacker). Attacker i physically closer to the verifier v than the prover p is. The first attack on **P1** described previously is an example for this scenario.

In this situation, we can show that **(a)** if an attacker can generate all the messages expected by the verifier from the *Request* to the *Response* without those messages being sent by the prover *and* **(b)** if all other messages expected by the verifier can also be generated by the attacker (with or without those messages emanating from the prover), then SDB is violated:

Theorem 1. *Suppose* $t_0, t_1, \ldots, t_m$ *are terms on* m *nodes on the verifier strand* v *with time of flight measured in between* t_0 *and* t_m*. Then, there exists a bundle with a violation of SDB if*

- *the constraints* $\langle t_1 : T_0, \ldots, t_m : T_m \rangle$ *are satisfiable where for* $i = 0$ *to* m*, every* $t \in T_i$ *either belongs to* T_0 *or a* $+$ *node on* v *and every* t_i *is a term on a* $-$ *node on* v*;*
- *all other* $-$ *nodes in* v *are realizable.*

Proof. Proof in [8].

Scenario 2 (farther attacker). Attacker i is physically farther from v than p. Here, i tries to show itself closer to v by using the responses from p to v in the DB part, and then inserts its own messages for the authentication part. **P1** is vulnerable in this scenario as well (Fig. 2.b).

This scenario is exactly opposite of Scenario 1: we just have to assume that the ideal bundle now is in between v and i instead of v and p. We should then analyze protocols for potential executions with p sending all the messages in the DB part and attacker sending the remaining messages. We prove this below:

Theorem 2. *Consider* v, p_1, p_2 *where* $d(v, p_1) < d(v, p_2)$*. Let* $\langle t_0, \ldots, t_m \rangle$ *be nodes on* v *between which time of flight is measured. Then, there is a violation of SDB if*

- *the constraints* $\langle t_1 : T_1, \ldots, t_m : T_m \rangle$ *are satisfiable where for all* $i = 1$ *to* m*, every* t_i *is unified with some* $t'_1 \in T_i$ *where* t'_i *is a term on* p_1*.*
- *All other '*$-$*' nodes of* v *are realizable without unifying with any subterms of* p_1*.*

Proof. Proof in [8].

6 Implementation and Examples

We now present some example protocols and their analyses using our technique. We tested all the protocols in the Constraint Solver tool with the scenarios and results in Section 5. We hosted all the protocols and scenarios in our on-line demo which can be tested with the click of a button. Here, we will present only the most interesting attacks and at least one per type of scenario.

It is worth mentioning that we made a simple change to the solver: we restricted it to consider only those node interleavings wherein the *Request* and *Response* nodes in the verifier strand immediately follow each other. We show in [8] that this is required to ensure soundness and that it preserves completeness wrt Def 13.

In all the protocols below, distance bound, $d = \frac{\delta_1 - \delta_2}{2} \times s$ verifier fixes δ_2 as a constant for a given protocol. Further to save space, we simplified some bundles by removing simple and insignificant relays.

6.1 P2 - Brands and Chaum [2]

The original Brands-Chaum protocol is a bit tricky with commit, rapid bit-level exchange and authentication/sign phases, and `XOR` operator that is not modeled by the solver. Hence, we analyzed an approximate version (Fig. 3.a).

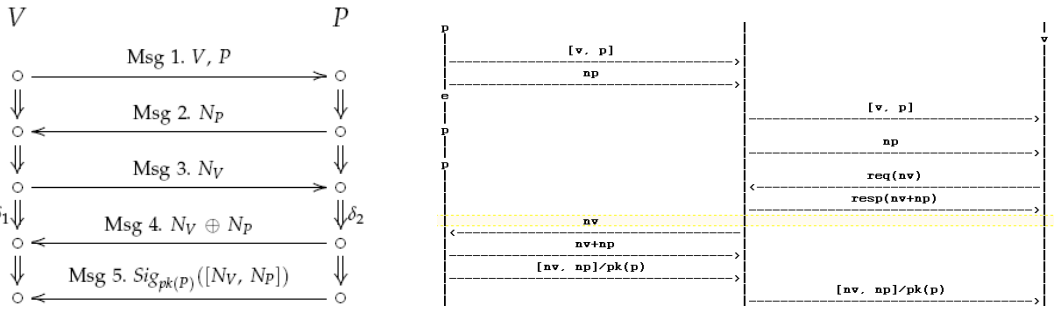


Fig. 3. (a) Brands-Chaum protocol **P2** (b) Actual solver trace of the MITM Attack on **P2** (Note: `nv+np` = $[n_v]^{\leftrightarrow}_{n_p}$, `[nv,np]/pk(p)` = $Sig_{pk(p)}[n_v, n_p]$; *Request* and *Response* nodes were coded as `req(nv)` and `resp(nv+np)`)

Notice that there is a pre-commitment of nonce N_P by P. Brands and Chaum specify that messages 3 and 4 should be bit-by-bit exchanges with the round-trip time calculated as the average of all the bit exchanges. Since the

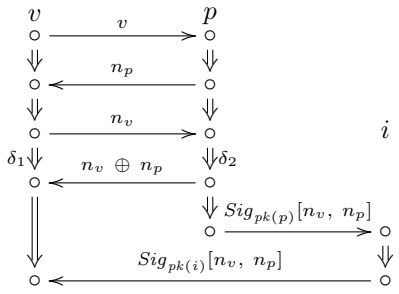
exchange is rapid and no other messages can interfere during the exchange, we felt it safe to model the protocol with just one of those message exchanges. Also, $N_V \oplus N_P$ was modeled as $[N_V]^{\leftrightarrow}_{N_P}$.

Honest prover, Closer attacker. Following our results in Section 5, we removed the nodes in the DB part in the prover strand and found an MITM attack on **P2** which was similar to the MITM attack on **P1** shown in the Introduction: Attacker simply sends all the messages except the signature to the verifier and later sends all of them to the prover. Finally, she relays the signature from the prover to the verifier. The solver found three different attack traces with three different node interleavings all essentially the same attack (Fig. 3.b).

The original Brands-Chaum protocol actually requires that the commitment N_P be secretly exchanged between V and P. With this requirement, the protocol forms a nice counter-example to Theorem 1: not all constraints corresponding to messages between Request (Msg 2) and Response (Msg 4) are satisfiable. When we made this change in the solver, it did not report an attack.

Dishonest prover, closer attacker. Obviously, revealing the nonce N_P (the commitment) to the attacker before hand allowed the attack (partial collusion) and of course, full collusion worked too. In any case, Brands-Chaum seems stronger against collusion than **P1** since it requires sharing of N_P for the attack to succeed.

Farther attacker. This protocol forms a nice example to test under Scenario 2. Assuming attacker is further away from the verifier, we followed our results in Section 5 and removed the nodes in the DB part in one strand while removing the signature (Msg 5) in another strand. The solver then output the following attack simplified by removing simple relays:



Here, i is further away from v than p and possibly out of a range that v wishes to include nodes. i then lets p respond to v's request, obstruct its authenticated response (Msg 5) and substitutes its own message signed with its own private key.

Obviously, this attack cannot work with the prover identity inside the signature, but only when the verifier uses the protocol to find its nearest neighbor.

6.2 P3 - Meadows et al. [10]

P3 below was recently proposed in [10] (Fig. 4.a).

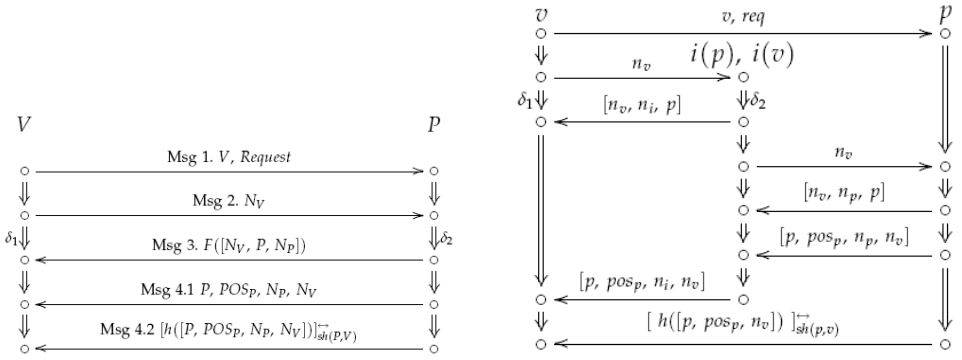


Fig. 4. (a) Meadows et al. protocol **P3** (b) MITM Attack

Honest prover, closer attacker. **P3** is actually quite similar to **P2** and Brands-Chaum but with some crucial changes. Even without any commitment step, it was not vulnerable to the MITM attack that we presented in Section 6.1 even though the nonce N_P is sent in plain in Msg 3 unlike Brands-Chaum that does not disclose it. This shows that sending N_P before the *Response* (Msg 3) was the fatal mistake in **P2**.

In any case, thus we believe that **P3** is stronger than **P2** and also the Brands-Chaum protocol since it does not require a previous set up to enable secure commit.

Dishonest prover, closer attacker. **P3** is vulnerable with partial collusion when i responds with Msg 2 and forwards n_v, and n_i to p later so that it can send the signature in Msg 5 to v with n_v, n_i, and other elements. However, p does not share any secrets with i to enable this attack. Hence, this protocol seems weaker than Brands-Chaum in this aspect.

Farther attacker. **P3** is also vulnerable to the "nearest-neighbor" attack that **P2** was, if we assume verifier does not know who it is talking to before receiving the signature in the final message. However, it would be unreasonable to make this assumption since the prover identity is explicitly included in the prior messages. Hence, we instantiated the prover variable P to a ground atomic value in the verifier strand when we tested this protocol, whence we could not reproduce the "nearest-neighbor" attack.

Tweaking P3. Since the protocol was resistant to all other scenarios except collusion, we tweaked with the protocol to appreciate the significance of individual elements and their placement in messages. We could not find the use or purpose of the field POS_P described anywhere in [10] but removing it did not reveal any new attack. It is interesting to ask if the nonce N_P inside Msg 4.2 is necessary. Removing it revealed an attack (Fig. 4.b).

6.3 P4 - Guttman et al. [5]

P4 differs from all others in having more than one encrypted message in the authentication part, seemingly extending the NSPK/NSL protocols (Fig. 5.a).

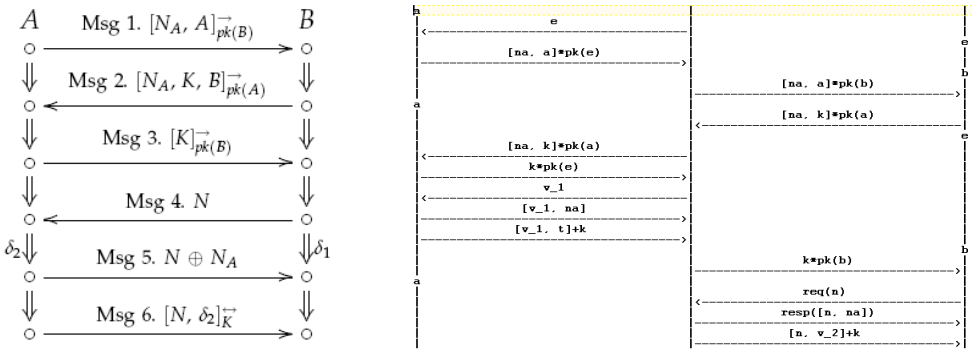


Fig. 5. (a) Guttman et al. protocol **P4** (b) Screen-shot of attack trace from the solver on the new Guttman et al.'s protocol. **Note**: v_1, v_2 are variables; `[t]*k` = $[t]_k^{\rightarrow}$, `[t]+k` = $[t]_k^{\leftrightarrow}$.

We analyzed this protocol with one strand per role in Scenarios A and 1; i.e. we considered an honest verifier B and an honest prover A with a MITM attacker who is physically in between them. Further, as usual, we tied the *Request* (Msg 4) and *Response* (Msg 5) together in the node interleaving. Without 'B' in Msg 2, the solver reported the trace with a MITM attack termed "Lowe style" attack in [5] (Fig. 5.b).

In the trace, the attacker plays MITM between a and b and learns k. Then, the *Response* $[n, n_a]$ is sent from the attacker's location, which is physically closer to the verifier b, violating SDB and also follows it up with an authentication of the challenge n in the last message $([n, \delta_2]_k^{\leftrightarrow})$.

The crux of this attack is the attacker's ability to satisfy both the conditions in Theorem 1. Satisfying the DB Part is trivial, but satisfying the authentication

part is possible only by breaking the secrecy of k since it is required to construct the last message, $[n,\ \delta_2]^{\leftrightarrow}_k$.

With larger semi-bundles/runs, more attacks could be possible by failing authentication even after the inclusion of 'B' in Msg 2; E.g., see attacks on NSL given in [11].

7 Conclusion

In this paper, we described a method to automatically analyze distance bounding protocols. We formalized the main property of secure distance bounding and explained how violations of it can be tested using the constraint solver. We also illustrated our technique by presenting analyses of some published protocols.

A natural extension to our work is to extend it to unbounded analysis since the constraint solver only considers bounded number of protocol processes. Unbounded verification tools such as ProVerif could be extended by tying the *Request* and *Rapid Response* together in the node interleavings as explained in Section 6, to produce attacks or to prove the absence of. In the case of ProVerif, this is as simple as adding four events in the protocol, two each for the verifier and prover in the protocol, corresponding to sending and receiving the *Request* and *Rapid Response* respectively. No other change in the tool is required.

Other areas for future work include extending our framework with multiple penetrators in the 3D space, analyzing other properties in this model such as denial of service, obtaining decidability results for distance bounding, and testing protocols with a more powerful solver that considers message operators with algebraic properties such as Exclusive-OR.

Recent related work. While the work in this paper was in progress, a related approach to verifying DB protocols using Isabelle/HOL was also in progress and is about to appear in [13]. We did a comparison of both:

- Being a verification effort, that approach differs from ours in the classical way that model checkers differ from theorem provers: the former tests for attacks while the latter proves the absence of. However, our approach can also be extended easily to unbounded verification, as explained above.
- Unlike many theorem provers and verifiers, our tool automatically considers on-line attack techniques such as type-flaws and messages from multiple protocols. In the past, the constraint solver was extended to find guessing attacks and was more powerful than other approaches due to these features [3]. Also, the solver is much faster than theorem provers, and answers usually in a fraction of a second, even when the vulnerability search fails.
- It was quite simple to extend the code for the solver and the protocols. In contrast, the authors in [13] report that formalization and proofs of a simple protocol such as **P3** (Meadows et al. [10]) took 13 pages (606 lines). Their complete formalizations took 136 pages of PDF documentation (7103 lines), while the constraint solver is 3 pages of Prolog of which we changed 4 lines. Our protocols were coded in 1 or at most 2 pages.

References

1. Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *IEEE Symposium on Security and Privacy*, pages 140–154, Oakland, California, May 2006.
2. S. Brands and D. Chaum. Distance-bounding protocols. In *Advances in Cryptology - EuroCrypt '93, LNCS 765*. Springer-Verlag, 1995.
3. R. Corin, S. Malladi, J. Alves-Foss, and S. Etalle. Guess what? Here is a new tool that finds some new guessing attacks. In *Workshop in the Issues of Theory of Security (WITS03), Poland, Warsaw*, April 2003.
4. Y. Desmedt. Major security problems with the ùnforgeable(́feige)-fiat-shamir proofs of identity and how to overcome them. In *SecureComm*, pages 15–17, SEDEP Paris, 1988.
5. J. D. Guttman, J. C. Herzog, V. Swarup, and F. J. Thayer. Strand spaces: From key exchange to secure location. In *Workshop on Event-based Semantics*, April 2008.
6. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055, pages 147–166. Springer-Verlag, 1996. Also in Software Concepts and Tools, 17:93-102, 1996.
7. Gavin Lowe. Some new attacks on cryptographic protocols. In *Proceedings of 9th Computer Security Foundations Workshop*. IEEE, 1996.
8. S. Malladi, B. Bruhadeshwar, and K. Kothapalli. Automatic analysis of distance bounding protocols. In *Technical Report, TR-Seed08*. Dakota State University, July 2009. Available at http://www.homepages.dsu.edu/malladis.
9. C. Meadows. Analyzing the Needham-Schroeder public-key protocol: A comparison of two approaches. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *ESORICS 96, LNCS 1146*, pages 351–364, 1996.
10. C. Meadows, R. Poovendran, D. Pavlovic, L. Chang, and P. Syverson. Distance bounding protocols: Authentication logic analysis and collusion attacks. In *Secure Localization and Time Synchronization for Wireless Sensor and Ad Hoc Networks*. Springer-Verlag, 2007.
11. J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. ACM Conference on Computer and Communication Security*, pages 166–175. ACM press, 2001.
12. N. Sastry, U. Shankar, and D. Wagner. Secure verification of location claims. In *ACM Workshop on Wireless security (WiSe 2003)*, pages 48–61. ACM, 2003.
13. P. Schaller, B. Schmidt, D. Basin, and S. Capkun. Modeling and verifying physical properties of security protocols for wireless networks. In *To Appear, Proc. 22nd Computer Security Foundations Workshop*. IEEE Computer Society Press, July 2009.
14. P. Syverson and C. Meadows. Dolev-Yao is no better than Machiavelli. In *Workshop in the Issues of Theory of Security, Poland, Warsaw*, 2000.
15. F. J. Thayer, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 160–171. IEEE Computer Society Press, 1998.